Problem Set 3—Due Friday, October 21

Problem 10. You are given an array of n processors, each of which can communicate with the processor immediately to its left (if present) and to its right (if present). The processors each have constant memory. The processors compute synchronously: there is a global clock that all can hear. Each processor i, for $1 \le i \le n$, initially holds an integer, x_i. Describe the most efficient protocol you can find to compute the median element among $x_1, \ldots, x_n$. All processors must learn this value. Count the exact number of cycles that your algorithm takes. You may assume that n is odd.

Problem 11. Problem 10.3-8, page 192.

Problem 12. In lecture we saw an analysis of the deterministic selection algorithm which treated the input array as columns of 5 objects, and we saw the resulting analysis yielded a linear time bound. Formulate the analogous algorithm using columns of 3 objects instead of 5, and give a bound on the running time of your algorithm. (*Hint: if you get a linear bound, you're doing something wrong.*)

Problem 13. You are given a box containing n ***nuts***, $N_1, \cdots, N_n$, and n ***bolts***, $B_1, \cdots, B_n$. Each nut fits one and only one bolt. You want to sort the nuts and bolts. Formally, you want to find the permutations π and π' such that $size(N_{\pi(1)}) < \cdots < size(N_{\pi(n)})$ and $size(B_{\pi'(1)}) < \cdots < size(B_{\pi'(n)})$. Thus nut $N_{\pi(j)}$ will fit bolt $B_{\pi'(j)}$ for each $1 \le j \le n$.

The only technique that you may use to assess the relative sizes of the nuts (and bolts) is to try to screw a particular nut N_i onto a particular bolt B_j. If the nut is too big for the bolt it will go on loosely, and you can tell that. If the nut is too small for the bolt it it won't go on at all, and you can tell that. And if the nut exactly fits the bolt, you can tell that, too. Thus the basic operation in our model of computation is $\circ(N_i, B_j) \in \{<, =, >\}$, which returns a value indicating that nut N_i is too small for bolt B_j, the right size for B_j, and too large for B_j, respectively.

- **(a)** Give an algorithm to solve the NUTS-AND-BOLTS problem in $O(n \lg n)$ *expected* time. Why does your algorithm run in the stated time bound? (*Hint: Think* QUICKSORT.)
- **(b)** Prove an $\Omega(n \lg n)$ lower bound for the worst case running time for any solution to NUTS-AND-BOLTS in the specified model of computation.

Problem 14. The MAJORITY Problem: given an unsorted array $A[1..n]$ of real numbers, find the one which occurs with multiplicity $> n/2$, or else report that there is no such number in the array.

- **(a)** An obvious method to solve this problem is to sort the reals in $\Theta(n \lg n)$ time. Give a linear-time algorithm which works by making a single call to a linear-time (deterministic) selection algorithm. How many comparisons does your algorithm use (in the worst case)?
- **(b)** Prove a lower bound of $\Omega(n)$ for the MAJORITY problem by using an adversary argument to show that every element of A must be examined.
- **(c)** Prove this fact: if $A[x] \neq A[y]$ then removing $A[x]$ and $A[y]$ from the array preserves the majority element, if it exists.
- **(d)** Give an example to show that the opposite of the fact above is false; give an array where there is no majority element but removal of two unequal members produces a majority element.
- **(e)** Using the results from parts (c) and (d), give an algorithm which efficiently solves the MAJORITY problem. It can be solved with only $2n$ comparisons, but it's acceptable if your algorithm uses up to $2n + o(n)$ comparisons.