HW 4 (due Wednesday, at noon, September 26, 2018)

CS 473: Algorithms, Fall 2018 Version: **1.1**

Submission guidelines and policies as in homework 1.

1 (100 PTS.) Sorting network by merging.

1.A. (10 PTS.) Prove that an n-input sorting network must contain at least one comparator between the ith and $(i+1)$th lines for all $i = 1, 2, ..., n-1$.

1.B. (20 PTS.) Suppose that we have $2n$ elements $< a_1, a_2, ..., a_{2n} >$ and wish to partition them into the n smallest and the n largest. Prove that we can do this in constant additional depth after separately sorting $< a_1, a_2, ..., a_n >$ and $< a_{n+1}, a_{n+2}, ..., a_{2n} >$.

(Hint: Go over the class notes for sorting networks – there is a three line argument here.).

1.C. (30 PTS.) Consider a merging network – such a network takes two sorted sequences and merge them into a sorted input. Specifically, in this case, the inputs are $a_1, a_2, \ldots, a_n$, for n an exact power of 2, in which the two monotonic sequences to be merged are $\langle a_1, a_3, \ldots, a_{n-1} \rangle$ and $\langle a_2, a_4, \ldots, a_n \rangle$ (namely, the input is a sequence of n numbers, where the odd numbers or sorted, and the even numbers are sorted). Prove that the number of comparators in this kind of merging network is $\Omega(n \log n)$. Why is this an interesting lower bound? (Hint: Partition the comparators into three sets: One involving inputs only the first half, inputs involving only the bottom half, and comparators involving both halves – then argue one of these sets has to have $\Omega(n)$ comparators, write down a recurrence, and solve it.)

1.D. (40 PTS.) Prove that any merging network, regardless of the order of inputs, requires $\Omega(n \log n)$ comparators. (Hint: Use part (C).)

2 (100 PTS.) Reorderings.

A *permutation network* on n inputs and n outputs has switches that allow it to connect its inputs to its outputs according to any $n!$ possible permutations. Figure 1 shows the 2-input, 2-output permutation network P_2, which consists of a single switch that can be set either to feed its inputs straight through to its outputs or to cross them.

2.A. (5 PTS.) Argue that if we replace each comparator in a sorting network with the switch of Figure 1, the resulting network is a permutation network. That is, for any permutation π, there is a way to set the switches in the network so that input i is connected to output $\pi(i)$.

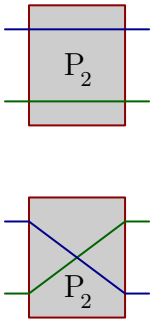

Figure 1: Permutation Switch

2.B. (2 PTS.) Figure 2 shows the recursive construction of an 8-input, 8-output permutation network P_8 that uses two copies of P_4 and 8 switches. The Switches have been set to realize the permutation $\pi = \langle \pi(1), \pi(2), \ldots, \pi(8) \rangle = \langle 4, 7, 3, 5, 1, 6, 8, 2 \rangle$, which requires (recursively) that the top P_4 realize $\langle 4, 2, 3, 1 \rangle$ and the bottom P_4 realize $\langle 2, 3, 1, 4 \rangle$.

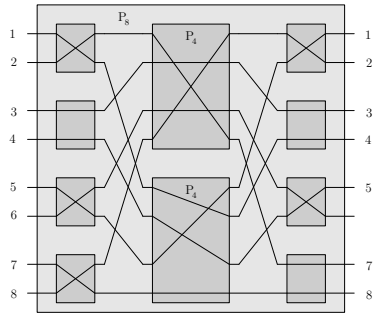

Figure 2: Permutation Network of Size 8

Show how to realize the permutation $\langle 5, 3, 4, 6, 1, 8, 2, 7\rangle$ on P_8 by drawing the switch settings and the permutations performed by the two P_4's.

2.C. (5 PTS.) Let n be an exact power of 2. Define P_n recursively in terms of two $P_{n/2}$'s in a manner similar to the way we defined P_8.

Describe an algorithm (ordinary random-access machine) that runs in $O(n)$-time that sets the n switches connected to the inputs and outputs of P_n and specifies the permutations that must be realized by each $P_{n/2}$ in order to accomplish any given n-element permutation. Prove that your algorithm is correct.

2.D. (3 PTS.) What are the depth size of P_n? How long does it take on an ordinary random-access machine to compute all switch settings, including those within the $P_{n/2}$'s?

2.E. (5 PTS.) Argue that for $n > 2$, any permutation network (not just P_n) must realize some permutation by two distinct combinations of switch settings.

3 (100 PTS.) Two is enough

Consider a uniform rooted tree of height h (every leaf is at distance h from the root). The root, as well as any internal node, has, say, 4 children. Each leaf has a boolean value associated with it. Each internal node returns true, if two or more of its children are one. The evaluation problem consists of determining the value of the root; at each step, an algorithm can choose one leaf whose value it wishes to read.

3.A. (35 PTS.) [This part is a challenging question – lets see what you can do.] Show that for any deterministic algorithm, there is an instance (a set of boolean values for the leaves) that forces it to read all $n = 4^h$ leaves. (Hint: Consider an adversary argument, where you provide the algorithm with the minimal amount of information as it requests bits from you. In particular, one can devise an adversary algorithm that forces you to read all input bits. And naturally, first solve the cases $h = 1$, $h = 2$, etc. Partial credit would be given for solving the cases $h = 1, 2, 3$.)

3.B. Consider a tree T with $h = 1$. Consider the algorithm that permute the children of the root of T, and evaluate them by this order. The algorithm stops as soon as one can compute the value of the root (for example, the algorithm encountered two ones). Let X be the number of leafs this algorithm evaluates. Prove that $\mathbb{E}[X] < 4$. What exactly is your upper bound for $\mathbb{E}[X]$?

(Hint: There are really only 5 different inputs: Calculate $\mathbb{E}[X]$ for each one of these scenarios, and return the weakest upper bound you found.)

3.C. Consider the recursive randomized algorithm that evaluates (recursively) the subtrees of a node in a random order, and stops as soon as the value of the node is resolved. (Unlike part (A), the specific input is fixed in advance – there is no adversary here.) Show the expected number of leaves read by the algorithm on any instance is at most n^c, where c is a constant strictly smaller than 1,