CS 473: Fundamental Algorithms, Fall 2011

Homework 3 (due Monday, 23:55:00, September 19, 2011)

Collaboration Policy & submission guidelines: See homework 1.
Each student individually have to also do **quiz 3** online.

Version: **1.0**

1. (30 PTS.) Fake coins.

   You are given a pile of n dollar coins: $C_1, \ldots, C_n$. You know that at most $n/2$ of the coins are fake. A fake coin has a different weight than a real coin. Two fake coins might have the same weight or a different weight. Unfortunately, the only operation you can do, called **compCoins**(C_i, C_j), is to compare the weight of two coins (C_i and C_j in this case), and return true if their weight is equal (it returns false otherwise). (A single call to **compCoins** takes constant time.)

   Describe an algorithm that finds all the real coins in the pile (i.e., it finds $n/2$, or more, coins such that all their weights are equal). What is the running time of your algorithm?

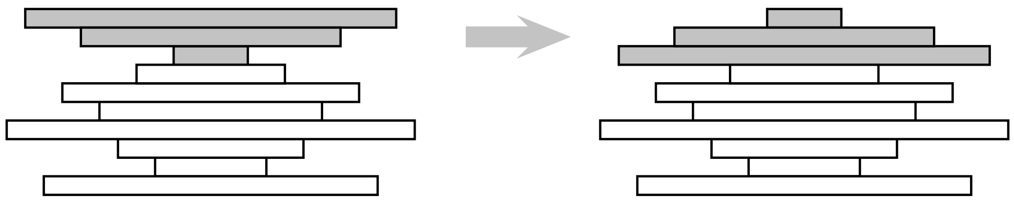
   The running time of your algorithm should be $O(n \log n)$ (a linear time algorithm here is possible but is considerably trickier). As a warm-up exercise think about $O(n^2)$ time algorithm.

   (You can assume that not all the fake coins have the same weight.)

   (A randomized algorithm for this problem is easy. To make things interesting, your algorithm has to be deterministic.)

2. (25 PTS.) Saving the world, one pancake at a time.

   Suppose we have a stack of n pancakes of different sizes. We want to sort the pancakes so that the smaller pancakes are on top of the larger pancakes. The only operation we can perform is a *flip* - insert a spatula under the top k pancakes, for some k between 1 and n, and flip them all over.

   (A) (15 PTS.) Describe an algorithm to sort an arbitrary stack of n pancakes and give a bound on the number of flips that the algorithm makes. Assume that the pancake information is given to you in the form of an n element array A. $A[i]$ is a number between 1 and n and $A[i] = j$ means that the j'th smallest pancake is in position i from the bottom; in other words $A[1]$ is the size of the bottom most pancake (relative to the others) and $A[n]$ is the size of the top pancake. Assume you have the operation Flip(k) which will flip the top k pancakes. Note that you are only interested in minimizing the number of flips.

(B) (10 PTS.) Suppose one side of each pancake is burned. Describe an algorithm that sorts the pancakes with the additional condition that the burned side of each pancake is on the bottom. Again, give a bound on the number of flips. In addition to A, assume that you have an array B that gives information on which side of the pancakes are burned; $B[i] = 0$ means that the bottom side of the pancake at the i'th position is burned and $B[i] = 1$ means the top side is burned. For simplicity, assume that whenever Flip(k) is done on A, the array B is automatically updated to reflect the information on the current pancakes in A.

3. (40 PTS.) Towers of Hanoi mess.

(A) (20 PTS.) A kid that was left unsupervised during to the visit to the temple of Hanoi, had messed up the Towers of Hanoi game completely. Now, the n disks are in two of the pegs (while the third peg is empty). Fortunately, the disks in each one of the two pegs are in the right order; that is, the disks are in a sorted order (bigger disks are below smaller disks). Provide an algorithm that takes this configuration, and figures out what sequence of operations needed to be done, so that all the disks are back to being in a single peg in a sorted order. As usual, you can move only a single disk at a time from one peg to another peg, and you can not place a bigger disk on top of a smaller disk. How many moves would your algorithm perform in the worst case (the smaller this bound, the better). (As usual, there are exactly three pegs.)

(B) (20 PTS.) (Less easy.) Another kid broke into the temple at night and played with the disks. When the monks woke up in the morning they discovered all the disks in a single peg but in a completely arbitrary order.
Describe an algorithm that is given this arbitrary configuration, and figures out what moves needs to be done, so that the disks are all in a single peg in the correct order. As before, you are not allowed to put a bigger disk on a smaller disk.
Prove that your algorithm works, and provide an upper bound on the number of moves it needs to do (the smaller the upper bound the better). As usual there are three pegs. (Hint: Use the algorithm from (A) as part of your algorithm.)