CS 473: Fundamental Algorithms, Fall 2014
HW 3 (due Tuesday, at noon, September 23, 2014)

Version: 1.01.

This homework contains three problems. **Read the instructions for submitting homework on the course webpage**.

Collaboration Policy: For this homework, Problems 1–3 can be worked in groups of up to three students.

Each student individually have to also do **quiz 3** online.

---

1. (30 PTS.) Backsquat Wars.

Once upon a time, everyone in Champaign, IL decided to become a Crossfit athlete and compete in the local Crossfit Games. There are several kinds of athletes. The coach decided that two athletes will be considered to be of the same kind if they can lift the same weight during back squat. Clearly, people of different strength cannot compete against each other, so the coach needs to decide how to separate the athletes into different categories. The only way to decide if two atheletes V_1 and V_2 are of the same kind, is to ask them to compete against each other by (**BackSquat**(V_1, V_2)) – if they lift different maximum weights, they are of different kinds, otherwise, if they lift the same maximum weight they are of the same kind. This check takes constant time. Specifically, **BackSquat**(V_1, V_2) returns true if V_1 and V_2 are of the same kind, and false otherwise.

As is usual in such cases, there is one kind that forms the majority of the Crossfit athletes. Since there is not enough space to hold a competition with all the athletes in Champaign participating, the Crossfit management decided that only athletes of the majority type will be competing. Given the n athletes $V_1, \ldots, V_n$ living in Champaign, describe an algorithm that uses the **BackSquat** operation and discovers all the athletes that belong to the majority type. Formally, $n \geq 4$, and there are at least 3 different kinds of athletes, and you have to discover all the athletes that belong to the kind that has at least $n/2$ members. You have to describe a deterministic algorithm that solves this problem in $O(n \log n)$ time (a faster algorithm is possible, but it is hard). An algorithm that runs in $O(n^2)$ time (or slower) would get at most 25% of the points, but you might want to first verify you know how to do it with this running time bound.

2. (40 PTS.) Liberty towers.

A German mathematician developed a new variant of the Towers of Hanoi game, known in the US literature as the "Liberty Towers" game[1]. Here, there are n disks placed on the first peg, and there are $k \geq 3$ pegs, numbered from 1 to k. You are allowed to move disks only on adjacent pegs (i.e., for peg i to peg $i+1$, or vice versa). Naturally, you are not allowed to put a bigger disk on a smaller disk. Your mission is to move all the disks from the first peg to the last peg.

[1] From Wikipedia: "During World War I, due to concerns the American public would reject a product with a German name, American sauerkraut makers relabeled their product as "Liberty cabbage" for the duration of the war."

(A) (10 PTS.) Describe a recursive algorithm for the case $k = 3$. How many moves does your algorithm do?

(B) (10 PTS.) Describe a recursive algorithm for the case $k = n + 1$. How many moves does your algorithm do? (The fewer, the better, naturally. For partial credit, the number of moves has to be at least polynomial in n.)

(C) (20 PTS.) Describe a recursive algorithm for the general case. How many moves does your algorithm do? (The fewer, the better, naturally.) In particular, how many moves does your algorithm do for $n = k^2$? A good solution should yield a solution that is as good as your solution for the (A) and (B) parts.

This question is somewhat more open ended – we have no specific solution at mind – do your best – but don't spend the rest of your life on this problem.

3. (30 PTS.) Selection.

You are given an array A with n distinct numbers in it, and another array B of ranks $i_1 < i_2 < \ldots < i_k$. An element x of A has rank u if there are exactly $u - 1$ numbers in A smaller than it. Design an algorithm that outputs the k elements in A that have the ranks $i_1, i_2, \ldots, i_k$.

(A) (25 PTS.) Describe a $O(n \log k)$ recursive algorithm for this problem. Prove the bound on the running time of the algorithm. (As a warm up first obtain an $O(nk)$ time algorithm).

(B) (5 PTS.) Show, that if this problem can be solved in $T(n, k)$ time, then one can sort n numbers in $O(n + T(n, n))$ time (i.e., give a reduction). What is an informal implication of this reduction in terms of being able to solve the problem in time faster than $O(n \log k)$ time.