HW 3 (due Monday, at noon, February 16, 2015)

OLD CS 473: Fundamental Algorithms, Spring 2015 Version: **1.02**

You also have to do the quiz online (on moodle).

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

---

1. (30 PTS.) Almost sorted.

Let A be an array of n *distinct* elements such that each element in A is at most k positions away from its position in the sorted order; here k is some integer such that $2 \leq k \leq n-1$. Thus the rank of an element at location $A[i]$ is at least $i-k$ and at most $i+k$; here the rank of an element is its position in the sorted array. In other words the array A is close to being sorted if k is small. Can we take advantage of this and sort faster? Design an $O(n \log k)$ time algorithm to sort elements of A. An $O(nk)$ algorithm will get you at most 10 points.

To make things easy assume that k is given to you (you can also think about the extension where k is not provided in advance).

2. (30 PTS.) Multi-median.

(A) (20 PTS.) You are given an array A with n real numbers in it[1]. You are also given k numbers, $i_1, \ldots, i_k$. Describe an algorithm that in $O(n \log k)$ time, outputs $e_{\text{rank}}(A, i_1), \ldots, e_{\text{rank}}(A, i_k)$, where $e_{\text{rank}}(A, t)$ is the tth smallest number in the array A.

(B) (10 PTS.) You are given an unsorted array A as above, and an integer number $t \geq 1$. Your algorithm should return any number in A that appears more than t times. If no number repeats itself t times, the algorithm should output that there is no such number. The running time of your algorithm should be $O(n \log(n/t))$.

Why no faster algorithm is possible for $t = 2$?

3. (40 PTS.) Towers of Annoy.

(A) (20 PTS.) In the *Towers of Saigon* game, the n disks are initially in two of the pegs (while the third peg is empty). The disks in each one of the two pegs are sorted.

Provide an algorithm that figures out what sequence of operations needed to be done, so that all the disks are in a single peg in a sorted order. One can move only a single disk at a time from one peg to another peg, and one 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).

(B) (20 PTS.) (Harder.) In the famous *Towers of Delhi* game, in the initial configuration, all the disks are in a single peg but in a completely arbitrary order (there are two additional empty pegs).

Describe an algorithm that computes a sequence of moves needed so that the disks are all in a single peg in the correct order. As before, one can **not** put a bigger disk on top of a smaller disk in the same peg.

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).

(Hint: Use the algorithm from (A) as part of your algorithm.)

---

[1] Real numbers, so you can not use hashing. Because, for example, floating point number representation is not unique, among other issues.