CS 473: Fundamental Algorithms, Spring 2014
HW 0 (due at noon on Tuesday, January 28, 2014)

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

You also have to do quiz 0 online!

Collaboration Policy: For this homework, each student should work independently and write up their own solutions and submit them.

Read the course policies before starting the homework.

- Homework 0 and Quiz 0 test your familiarity with prerequisite material: big-Oh notation, elementary algorithms and data structures, recurrences, graphs, and most importantly, induction, to help you identify gaps in your background knowledge. You are responsible for filling those gaps. The course web page has pointers to several excellent online resources for prerequisite material. If you need help, please ask in headbanging, on Piazza, in office hours, or by email.
- Each student must submit individual solutions for these homework problems. For all future homeworks, groups of up to three students may submit a single group solution for each problem.
- Please carefully read the course policies on the course web site. If you have any questions, please ask in lecture, in headbanging, on Piazza, in office hours, or by email. In particular:
  - Submit separately stapled solutions, one for each numbered problem, with your name and NetID clearly printed on each page.
  - You may use any source at your disposal: paper, electronic, human, or other, but you must write your solutions in your own words, and you must cite every source that you use (except for official course materials). Please see the academic integrity policy for more details.
  - No late homework will be accepted for any reason. However, we may forgive quizzes or homeworks in extenuating circumstances.
  - Answering "I don't know" to any (non-extra-credit) problem or subproblem, on any homework or exam, is worth 25% partial credit.
  - Algorithms or proofs containing phrases like and so on or repeat this process for all n, instead of an explicit loop, recursion, or induction, will receive a score of 0.
  - Unless explicitly stated otherwise, every homework problem requires a proof.

---

1 Required problems

1. (50 PTS.) Intersecting trees

Suppose T is a tree and $T_1, T_2, \ldots, T_k$ are subtrees of T such that for any $1 \leq i < j \leq k$ the trees T_i and T_j intersect (they share a node). Then, show that the trees $T_1, T_2, \ldots, T_k$ intersect at a common node; that is, there is a node $v \in V$ that is contained in T_i for all $1 \leq i \leq k$. Now consider the case when $T_1, T_2, \ldots, T_k$ are trees that are subgraphs of a connected graph G. Is it still true that if the subtrees intersect pairwise then they have a common intersection? Prove or disprove via a counter example.

2. (50 PTS.) Priority Queues

A priority queue data structure stores items of the form (a, p) where a is some object and p is a priority or key from some totally ordered universe (we will assume for simplicity that p is a number). You have seen implementations of priority queues where the operations such as insert, delete, find-min, and extract-min take $O(\log n)$ time where n is the number of items stored in the data structure at the time of the operation. In some situations, we can guarantee that the number of *distinct* priorities of the items stored in the data structure will be no more than some given bound k, that is independent of the number of items stored in the data structure. Describe an implementation of priority queues that has an $O(\log k)$ worst-case bound on the time to implement insert and extract-min and $O(1)$ for find-min. Extract-min can output any item with the smallest priority among the items stored in the data structure. Your data structure should have the following additional robustness. If the user inserts items with more than k distinct priorities then it should detect and report that. Note that the number of items n can be much larger than k. Moreover you are not allowed to assume that the priorities are from a static set of k numbers, only that at any time, the number of distinct priorities in the items stored in the data structure is at most k. A formal proof of correctness for your implementation is not required. If you use hashing-based data structures you should rigorously define how you would guarantee a worst-case bound on the running time.