CS 466/666, Spring 2015
Timothy Chan

Assignment 2 (due June 10 Wednesday 5pm)

Please read `http://www.student.cs.uwaterloo.ca/~cs466/policies.html` first for general instructions.

1. [*15 marks*] We want a data structure that supports the following operations on a collection of *not necessarily disjoint* sets of elements (real numbers):

   - makeset(x): create a set containing one element of value x (there may be other elements having the same value).
   - union(S_1, S_2): create a set $S_1 \cup S_2$, with duplicates removed (S_1 and S_2 may share common values). The new set is now in the collection, but the old sets S_1 and S_2 are not.
   - size(S): return the number of distinct values in the set S.

   Describe a solution that achieves $O(\log^2 n)$ amortized time per operation, where n is the number of makesets.

   [Hint: we can't directly apply a union-find data structure here, but the weighted-union heuristic can still help. Instead of linked lists, store each set in a balanced search tree. For union, use repeated insertions to one of the search trees...]

2. [*15 marks*] Give a data structure to support the following operations on a set S of intervals in one dimension:

   - insert(a, b): given two integers $a, b \in \{1, 2, \ldots, U\}$, insert the interval $[a, b]$ to S.
   - query(x): given integer $x \in \{1, 2, \ldots, U\}$, return yes iff x lies in the union of the intervals in S.

   (For example, if S contains the intervals $[1, 4]$, $[6, 10]$, $[8, 13]$, the union of S is $[1, 4] \cup [6, 13]$.)

   Your solution should have $O(\alpha(U))$ amortized insert and query time under the assumption that the number n of inserts in the sequence is at least $\Theta(U)$.

   [Hint: as you might guess, you should use the union-find data structure from class as a subroutine. Over a sequence of n inserts, how many calls to union does your algorithm make?]

3. [*10 marks*] In a popular form of logic puzzles, you land on an island that has three types of inhabitants: "knights", who always tell the truth; "knaves", who always lie; and "spies", who sometimes lie and sometimes tell the truth.

   Suppose there are n inhabitants, where 60% are known to be decent folks, i.e., knights. The remaining 40% are bad, i.e., knaves or spies. You want to know who are the good/bad guys,

i.e., determine the types of all n inhabitants. You are allowed to ask only questions of the form, "is person A a knight/knave/spy?", to another person B. (All the inhabitants know each other.) Obviously, if you can find a person who you know is a knight, the problem is solved after asking n additional questions.

(a) [*2 marks*] Give a (very) efficient Monte-Carlo algorithm that finds a knight. State the probability of error. [Hint: this is supposed to be easy!]

(b) [*8 marks*] Give a Las-Vegas algorithm that finds a knight by asking $O(n)$ expected number of questions. Analyze the constant factor in the big-Oh and make it smaller than 1.5. [Hint: use (a). How can you confirm whether a specific person is a knight by asking $O(n)$ questions?]

[Note: there is also a deterministic algorithm that requires $O(n)$ questions, but it is more complicated and has a larger constant.]