CS 473 ✧ Spring 2016

Homework 3

Due Tuesday, February 9, 2016, at 8pm

Unless a problem specifically states otherwise, you may assume a function RANDOM that takes a positive integer k as input and returns an integer chosen uniformly and independently at random from $\{1,2,\ldots,k\}$ in $O(1)$ time. For example, to flip a fair coin, you could call RANDOM(2).

---

1. Suppose we want to write an efficient function RANDOMPERMUTATION(n) that returns a permutation of the set $\{1,2,\ldots,n\}$ chosen uniformly at random.

   (a) Prove that the following algorithm is ***not*** correct. *[Hint: There is a one-line proof!]*

   ```
   RandomPermutation(n):
     for i ← 1 to n
         π[i] ← i
     for i ← 1 to n
         swap π[i] ↔ π[Random(n)]
   ```

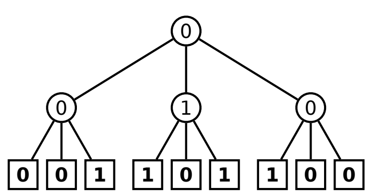
   (b) Consider the following implementation of RANDOMPERMUTATION.

   ```
   RandomPermutation(n):
     for i ← 1 to n
         π[i] ← NULL
     for i ← 1 to n
         j ← Random(n)
         while (π[j] != NULL)
             j ← Random(n)
         π[j] ← i
     return π
   ```

   Prove that this algorithm is correct and analyze its expected running time.

   (c) Describe and analyze an implementation of RANDOMPERMUTATION that runs in expected worst-case time $O(n)$.

2. A ***majority tree*** is a complete ternary tree in which every leaf is labeled either 0 or 1. The *value* of a leaf is its label; the *value* of any internal node is the majority of the values of its three children. For example, if the tree has depth 2 and its leaves are labeled $1,0,0,0,1,0,1,1,1$, the root has value 0.



A majority tree with depth 2.

It is easy to compute value of the root of a majority tree of depth n in $O(3^n)$ time, given the sequence of 3^n leaf labels as input, using a simple post-order traversal of the tree. Prove that this simple algorithm is optimal, and then describe a better algorithm. More formally:

(a) Prove that *any* deterministic algorithm that computes the value of the root of a majority tree *must* examine every leaf. *[Hint: Consider the special case $n = 1$. Recurse.]*

(b) Describe and analyze a randomized algorithm that computes the value of the root in worst-case expected time $O(c^n)$ for some explicit constant $c < 3$. *[Hint: Consider the special case $n = 1$. Recurse.]*

3. A ***meldable priority queue*** stores a set of keys from some totally-ordered universe (such as the integers) and supports the following operations:

- MakeQueue: Return a new priority queue containing the empty set.
- FindMin(Q): Return the smallest element of Q (if any).
- DeleteMin(Q): Remove the smallest element in Q (if any).
- Insert(Q, x): Insert element x into Q, if it is not already there.
- DecreaseKey(Q, x, y): Replace an element $x \in Q$ with a smaller key y. (If $y > x$, the operation fails.) The input is a pointer directly to the node in Q containing x.
- Delete(Q, x): Delete the element $x \in Q$. The input is a pointer directly to the node in Q containing x.
- Meld(Q_1, Q_2): Return a new priority queue containing all the elements of Q_1 and Q_2; this operation destroys Q_1 and Q_2.

A simple way to implement such a data structure is to use a heap-ordered binary tree, where each node stores a key, along with pointers to its parent and two children. Meld can be implemented using the following randomized algorithm:

```
Meld(Q1, Q2):
  if Q1 is empty return Q2
  if Q2 is empty return Q1
  if key(Q1) > key(Q2)
      swap Q1 ↔ Q2
  with probability 1/2
      left(Q1) ← Meld(left(Q1), Q2)
  else
      right(Q1) ← Meld(right(Q1), Q2)
  return Q1
```

(a) Prove that for *any* heap-ordered binary trees Q_1 and Q_2 (not just those constructed by the operations listed above), the expected running time of Meld(Q_1, Q_2) is $O(\log n)$, where $n = |Q_1| + |Q_2|$. *[Hint: What is the expected length of a random root-to-leaf path in an n-node binary tree, where each left/right choice is made with equal probability?]*

(b) Prove that Meld(Q_1, Q_2) runs in $O(\log n)$ time with high probability.

(c) Show that each of the other meldable priority queue operations can be implemented with at most one call to Meld and $O(1)$ additional time. (It follows that each operation takes only $O(\log n)$ time with high probability.)