CS 473G: Combinatorial Algorithms, Fall 2005
Homework 4

Due Thursday, October 27, 2005, at midnight

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

Homeworks may be done in teams of up to three people. Each team turns in just one solution; every member of a team gets the same grade.

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Staple this sheet to the top of your solution to problem 1.

If you are an I2CS student, print "(I2CS)" next to your name. Teams that include both on-campus and I2CS students can have up to four members. Any team containing both on-campus and I2CS students automatically receives 3 points of extra credit.

> For the rest of the semester, unless specifically stated otherwise, you may assume that the function RANDOM(m) returns an integer chosen uniformly at random from the set $\{1, 2, , \ldots, m\}$ in $O(1)$ time. For example, a fair coin flip is obtained by calling RANDOM(2).

1. Consider the following randomized algorithm for choosing the largest bolt. Draw a bolt uniformly at random from the set of n bolts, and draw a nut uniformly at random from the set of n nuts. If the bolt is smaller than the nut, discard the bolt, draw a new bolt uniformly at random from the unchosen bolts, and repeat. Otherwise, discard the nut, draw a new nut uniformly at random from the unchosen nuts, and repeat. Stop either when every nut has been discarded, or every bolt except the one in your hand has been discarded.

   What is the *exact* expected number of nut-bolt tests performed by this algorithm? Prove your answer is correct. [Hint: What is the expected number of unchosen nuts and bolts when the algorithm terminates?]

2. 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(Q_1, Q_2):
   >   if Q_1 is empty return Q_2
   >   if Q_2 is empty return Q_1
   >
   >   if $key(Q_1) > key(Q_2)$
   >   swap $Q_1 \leftrightarrow Q_2$
   >
   >   with probability 1/2
   > $left(Q_1) \leftarrow$ MELD($left(Q_1), Q_2$)
   >   else
   > $right(Q_1) \leftarrow$ MELD($right(Q_1), Q_2$)
   >
   >   return Q_1

   (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: How long is a random root-to-leaf path in an n-node binary tree if 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 cab be implemented with at most one call to MELD and $O(1)$ additional time. (This implies that every operation takes $O(\log n)$ time with high probability.)

3. Let $M[1\mathbin{..}n][1\mathbin{..}n]$ be an $n \times n$ matrix in which every row and every column is sorted. Such an array is called *totally monotone*. No two elements of M are equal.

   (a) Describe and analyze an algorithm to solve the following problem in $O(n)$ time: Given indices i, j, i', j' as input, compute the number of elements of M smaller than $M[i][j]$ and larger than $M[i'][j']$.

   (b) Describe and analyze an algorithm to solve the following problem in $O(n)$ time: Given indices i, j, i', j' as input, return an element of M chosen uniformly at random from the elements smaller than $M[i][j]$ and larger than $M[i'][j']$. Assume the requested range is always non-empty.

   (c) Describe and analyze a randomized algorithm to compute the median element of M in $O(n \log n)$ expected time.

4. Let $X[1\mathbin{..}n]$ be an array of n distinct real numbers, and let $N[1\mathbin{..}n]$ be an array of indices with the following property: If $X[i]$ is the largest element of X, then $X[N[i]]$ is the smallest element of X; otherwise, $X[N[i]]$ is the smallest element of X that is larger than $X[i]$.

   For example:

i	1	2	3	4	5	6	7	8	9
$X[i]$	83	54	16	31	45	99	78	62	27
$N[i]$	6	8	9	5	2	3	1	7	4

   Describe and analyze a randomized algorithm that determines whether a given number x appears in the array X in $O(\sqrt{n})$ expected time. **Your algorithm may not modify the arrays X and $Next$.**

5. A *majority tree* is a complete ternary tree with depth n, where 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. Consider the problem of computing the value of the root of a majority tree, given the sequence of 3^n leaf labels as input. For example, if $n = 2$ and the leaves are labeled $0, 0, 1, 1, 0, 1, 1, 1, 0, 0$, the root has value 0.
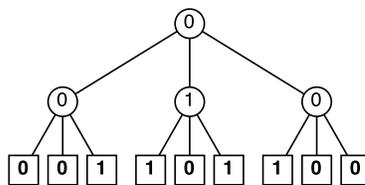
A majority tree with depth $n = 2$.

   (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 constant $c < 3$. [Hint: Consider the special case $n = 1$. Recurse.]

*6. *[Extra credit]* In the usual theoretical presentation of treaps, the priorities are random real numbers chosen uniformly from the interval $[0, 1]$, but in practice, computers only have access to random *bits*. This problem asks you to analyze a modification of treaps that takes this limitation into account.

Suppose the priority of a node v is abstractly represented as an infinite sequence $\pi_v[1 .. \infty]$ of random bits, which is interpreted as the rational number

$$priority(v) = \sum_{i=1}^{\infty} \pi_v[i] \cdot 2^{-i}.$$

However, only a finite number ℓ_v of these bits are actually known at any given time. When a node v is first created, *none* of the priority bits are known: $\ell_v = 0$. We generate (or 'reveal') new random bits only when they are necessary to compare priorities. The following algorithm compares the priorities of any two nodes in $O(1)$ expected time:

LARGERPRIORITY(v, w):
    for $i \leftarrow 1$ to ∞
        if $i > \ell_v$
            $\ell_v \leftarrow i$; $\pi_v[i] \leftarrow$ RANDOMBIT
        if $i > \ell_w$
            $\ell_w \leftarrow i$; $\pi_w[i] \leftarrow$ RANDOMBIT
        if $\pi_v[i] > \pi_w[i]$
            return v
        else if $\pi_v[i] < \pi_w[i]$
            return w

Suppose we insert n items one at a time into an initially empty treap. Let $L = \sum_v \ell_v$ denote the total number of random bits generated by calls to LARGERPRIORITY during these insertions.

  (a) Prove that $E[L] = \Theta(n)$.

  (b) Prove that $E[\ell_v] = \Theta(1)$ for any node v. *[Hint: This is equivalent to part (a). Why?]*

  (c) Prove that $E[\ell_{\text{root}}] = \Theta(\log n)$. *[Hint: Why doesn't this contradict part (b)?]*