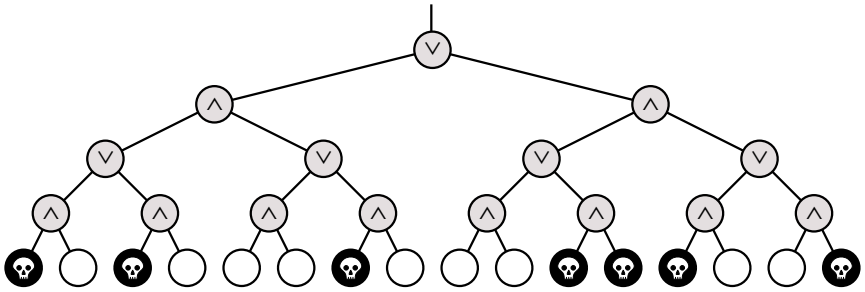
New CS 473: Algorithms, Spring 2015
Homework 6

Due Tuesday, March 17, 2015 at 5pm

All homework must be submitted electronically via Moodle as separate PDF files, one for each numbered problem. Please see the course web site for more information.

---

1. Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with 4^n leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after $2n$ moves, when the token lands on a leaf. If the final leaf is black, you die; if it's white, you will live forever. You move first, so Death gets the last turn.



You can decide whether it's worth playing or not as follows. Imagine that the nodes at even levels (where it's your turn) are OR gates, the nodes at odd levels (where it's Death's turn) are AND gates. Each gate gets its input from its children and passes its output to its parent. White and black stand for TRUE and FALSE. If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should challenge Death to a game of Twister instead.

(a) Describe and analyze a deterministic algorithm to determine whether or not you can win. *[Hint: This is easy, but be specific!]*

*(b) ***[Extra credit]*** Prove that *any* deterministic algorithm that correctly determines whether you can win *must* examine every leaf in the tree. It follows that any correct algorithm for part (a) *must* take $\Omega(4^n)$ time. *[Hint: Let Death cheat, but not in a way that the algorithm can detect.]*

(c) Unfortunately, Death won't give you enough time to look at every node in the tree. Describe a *randomized* algorithm that determines whether you can win in $O(3^n)$ expected time. *[Hint: Consider the case $n = 1$.]*

*(d) ***[Extra credit]*** Describe and analyze a randomized algorithm that determines whether you can win in $O(c^n)$ expected time, for some explicit constant $c < 3$. Your analysis should yield an exact value for the constant c. *[Hint: You may not need to change your algorithm from part (b) at all!]*

2. A ***meldable priority queue*** stores a set of priorities 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.
   - DECREASEPRIORITY(Q, x, y): Replace an element $x \in Q$ with a new element $y < x$. (If $y \geq x$, the operation fails.) The input includes 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. The elements of Q_1 and Q_2 could be arbitrarily intermixed; we do *not* assume, for example, that every element of Q_1 is smaller than every element of Q_2.

   A simple way to implement such a data structure is to use a heap-ordered binary tree, where each node stores a priority, along with pointers to its parent and two children. MELD can be implemented using the following randomized algorithm. The input consists of pointers to the roots of the two trees.

```
MELD(Q1, Q2):
  if Q1 = NULL then return Q2
  if Q2 = NULL then return Q1

  if priority(Q1) > priority(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. *[Hint: You can use Chernoff bounds, but the simpler identity $\binom{c}{k} \leq (ce)^k$ is actually sufficient.]*

   (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 every operation takes $O(\log n)$ time with high probability.)

New CS 473 Spring 2015 — Homework 6 Problem 1

Name:	NetID:
Name:	NetID:
Name:	NetID:

(a) Describe and analyze a deterministic algorithm to determine whether or not you can win the Death game.

(b) ***[Extra credit]*** Prove that *any* deterministic algorithm that correctly determines whether you can win *must* examine every leaf of the input tree.

(c) Describe a *randomized* algorithm that determines whether you can win the Death game in $O(3^n)$ expected time.

(d) ***[Extra credit]*** Describe and analyze a randomized algorithm that determines whether you can win the Death game in $O(c^n)$ expected time, for some constant $c < 3$. Your analysis should yield an exact value for the constant c.

New CS 473 Spring 2015 — Homework 6 Problem 2

Name:	NetID:
Name:	NetID:
Name:	NetID:

(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 $\textsc{Meld}(Q_1, Q_2)$ is $O(\log n)$, where $n = |Q_1| + |Q_2|$.

(b) Prove that $\textsc{Meld}(Q_1, Q_2)$ 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 $\textsc{Meld}$ and $O(1)$ additional time.