

# CS 473: Undergraduate Algorithms, Spring 2009

## Homework 3

Written solutions due Tuesday, March 2, 2009 at 11:59:59pm.

---

1. 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
     $left(Q_1) \leftarrow \text{MELD}(left(Q_1), Q_2)$ 
  else
     $right(Q_1) \leftarrow \text{MELD}(right(Q_1), Q_2)$ 
  return  $Q_1$ 

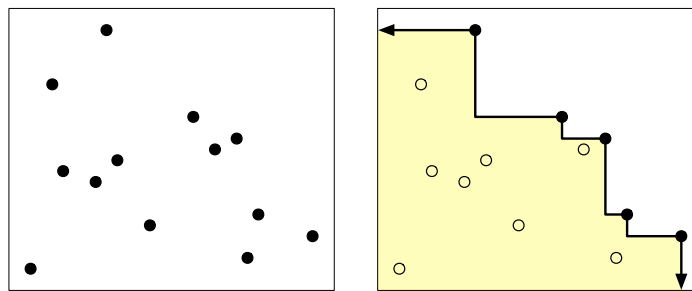
```

- (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  $n$  is the total number of nodes in both trees. [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) 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. (This implies that every operation takes  $O(\log n)$  expected time.)

2. Recall that a *priority search tree* is a binary tree in which every node has both a *search key* and a *priority*, arranged so that the tree is simultaneously a binary search tree for the keys and a min-heap for the priorities. A *heater* is a priority search tree in which the priorities are given by the user, and the search keys are distributed uniformly and independently at random in the real interval  $[0, 1]$ . Intuitively, a heater is the ‘opposite’ of a treap.

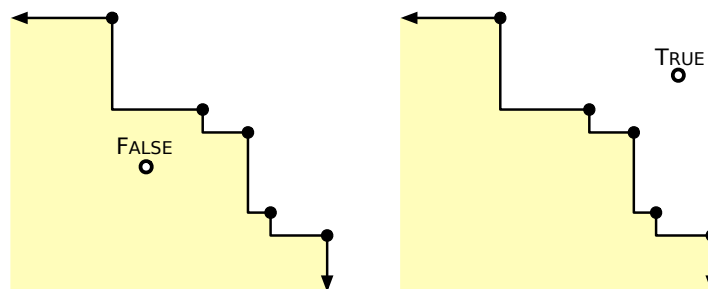
The following problems consider an  $n$ -node heater  $T$  whose node priorities are the integers from 1 to  $n$ . We identify nodes in  $T$  by their priorities; thus, ‘node 5’ means the node in  $T$  with priority 5. The min-heap property implies that node 1 is the root of  $T$ . Finally, let  $i$  and  $j$  be integers with  $1 \leq i < j \leq n$ .

- (a) **Prove** that in a random permutation of the  $(i + 1)$ -element set  $\{1, 2, \dots, i, j\}$ , elements  $i$  and  $j$  are adjacent with probability  $2/(i + 1)$ .
  - (b) **Prove** that node  $i$  is an ancestor of node  $j$  with probability  $2/(i + 1)$ . [Hint: Use part (a)!]
  - (c) What is the probability that node  $i$  is a *descendant* of node  $j$ ? [Hint: **Don’t** use part (a)!]
  - (d) What is the *exact* expected depth of node  $j$ ?
3. Let  $P$  be a set of  $n$  points in the plane. The *staircase* of  $P$  is the set of all points in the plane that have at least one point in  $P$  both above and to the right.



A set of points in the plane and its staircase (shaded).

- (a) Describe an algorithm to compute the staircase of a set of  $n$  points in  $O(n \log n)$  time.
- (b) Describe and analyze a data structure that stores the staircase of a set of points, and an algorithm `ABOVE?(x, y)` that returns `TRUE` if the point  $(x, y)$  is above the staircase, or `FALSE` otherwise. Your data structure should use  $O(n)$  space, and your `ABOVE?` algorithm should run in  $O(\log n)$  time.



Two staircase queries.