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

- **MAKEQUE**: 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 — each node stores a priority, which is smaller than the priorities of its children, along with pointers to its parent and at most two children. **MELD** can be implemented using the following randomized algorithm:

\[
\text{MELD}(Q_1, Q_2):
\begin{align*}
\text{if } Q_1 \text{ is empty return } Q_2 \\
\text{if } Q_2 \text{ is empty return } Q_1 \\
\text{if } \text{priority}(Q_1) > \text{priority}(Q_2) \\
\hspace{1cm} \text{swap } Q_1 \leftrightarrow Q_2 \\
\hspace{1cm} \text{with probability } 1/2 \\
\hspace{2cm} \left(Q_1 \leftarrow \text{MELD}(\text{left}(Q_1), Q_2) \right) \\
\hspace{1cm} \text{else} \\
\hspace{2cm} \left(Q_1 \leftarrow \text{MELD}(\text{right}(Q_1), Q_2) \right) \\
\text{return } Q_1
\end{align*}
\]

(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** 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 uniformly and independently at random?]

(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 treap is a priority search tree whose search keys are given by the user and whose priorities are independent random numbers.

A heater is a priority search tree whose priorities are given by the user and whose search keys are distributed uniformly and independently at random in the real interval \([0, 1]\). Intuitively, a heater is a sort of anti-treap.\(^1\)

\(^1\)There are those who think that life has nothing left to chance, a host of holy horrors to direct our aimless dance.
The following problems consider an n-node heater $T$. We identify nodes in $T$ by their priority rank; for example, “node 5” means the node in $T$ with the 5th smallest priority. The min-heap property implies that node 1 is the root of $T$. You may assume all search keys and priorities are distinct. Finally, let $i$ and $j$ be arbitrary integers with $1 \leq i < j \leq n$.

(a) Prove that if we permute the set $\{1, 2, \ldots, n\}$ uniformly at random, integers $i$ and $j$ are adjacent with probability $2/n$.

(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 $j$?

(e) Describe and analyze an algorithm to insert a new item into an $n$-node heater.

(f) Describe and analyze an algorithm to delete the smallest priority (the root) from an $n$-node heater.

*3. Extra credit; due October 15. In the usual theoretical presentation of treaps, the priorities are random real numbers chosen uniformly from the interval $[0, 1]$. In practice, however, computers have access only to random bits. This problem asks you to analyze an implementation 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

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

However, only a finite number $\ell_v$ 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:

```
LARGER_PRIORITY(v, w):
for i ← 1 to ∞
    if i > \ell_v
        \ell_v ← i; \pi_v[i] ← RANDOM_BIT
    if i > \ell_w
        \ell_w ← i; \pi_w[i] ← RANDOM_BIT
    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 LARGER_PRIORITY 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)?]
(a) Prove that for any heap-ordered binary trees $Q_1$ and $Q_2$, the expected running time of $\text{MELD}(Q_1, Q_2)$ is $O(\log n)$, where $n = |Q_1| + |Q_2|$.

(b) Show that each of the other meldable priority queue operations can be implemented with at most one call to $\text{MELD}$ and $O(1)$ additional time.
(a) Prove that if we permute the set \{1, 2, \ldots, n\} uniformly at random, integers \( i \) and \( j \) are adjacent with probability \( 2/n \).

(b) Prove that in any heater, node \( i \) is an ancestor of node \( j \) with probability \( 2/(i + 1) \).

(c) What is the probability that node \( i \) is a descendant of node \( j \)?

(d) What is the exact expected depth of node \( j \)?

(e) Describe and analyze an algorithm to insert a new item into a heater.

(f) Describe and analyze an algorithm to delete the smallest priority (the root) from a heater.
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 \( v \).

(c) Prove that \( E[\ell_{\text{root}}] = \Theta(\log n) \).