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 uniformly and independently at random?]*

Solution: Consider merging two arbitrary heap-ordered binary trees Q_1 and Q_2. The execution of the MELD algorithm traces random paths downward from the roots of Q_1 and Q_2. For example, the path in Q_1 is extended downward, with equal probability either left or right, when the current node in Q_1 has lower priority than the current node in Q_2. Thus, the running time of MELD(Q_1, Q_2) is *at most* $P(Q_1) + P(Q_2)$, where $P(T)$ denotes the length of a random root-to-leaf path in binary tree T.

To complete the analysis, we need to prove that $\mathrm{E}[P(Q_1) + P(Q_2)] = O(\log n)$, where n is the total number of nodes in both trees. It suffices to prove that for every binary tree T with *at most* n nodes, we have $P(T) = O(\log n)$.

So let $P(n)$ denote the maximum, over all trees T with at most n nodes, of $P(T)$. We claim that $E[P(n)] \le \lg(n+1)$. If $n = 0$, this claim is vacuously true, because there are no trees with at most 0 nodes. Otherwise, let T be an arbitrary binary tree with at most n nodes. Suppose the left and right subtrees of T contain ℓ and r nodes, respectively; observe that $\ell + r + 1 \le n$. The inductive hypothesis implies that $\mathrm{E}[P(\ell)] \le \lg(\ell + 1)$ and $\mathrm{E}[P(r)] \le \lg(r+1)$. Thus, the expected length of a random root-to-leaf path in T is at most

$$\begin{aligned}
\frac{1}{2}\Big(\mathrm{E}[P(\ell)] + \mathrm{E}[P(r)]\Big) &\le \frac{1}{2}\Big(\lg(\ell+1) + \lg(r+1)\Big) \\
&= \lg\left(2\sqrt{(\ell+1)(r+1)}\right) \\
&\le \lg\Big((\ell+1) + (r+1)\Big) \\
&\le \lg(n+1).
\end{aligned}$$

(The second-to-last step uses the inequality $2\sqrt{xy} \le x + y$, which follows immediately from the fact that $(\sqrt{x} - \sqrt{y})^2 \ge 0$.) ■

Rubric: 5 points max: 3 points for proving expected path length is $O(\log n)$ + 2 points for analyzing MELD

(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.

Solution:

- FINDMIN(Q): Return $key(Q)$
- DELETEMIN(Q): Return MELD$(left(Q), right(Q))$
- INSERT(Q, x): Create a new one-node priority queue Q_x containing x, and then return MELD(Q, Q_x).
- DECREASEKEY(Q, x, y) : Let Q_x be the node containing x. Detach Q_x from its parent, set $key(Q_x) \leftarrow y$, and return MELD(Q, Q_x).
- DELETE(Q, x): Let Q_x be the the node containing x. Replace Q_x with DELETEMIN(Q_x) (which calls MELD$(left(Q_x), right(Q_x))$) and return Q.

■

Rubric: 5 points max: 1 point each. These are not the only correct solutions.

2. 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 $1 \le i < j \le n$.

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

   Solution: Fix a permutation of the $(n-1)$-element set $\{1, 2, \dots, n\} \setminus \{j\}$. There are exactly $i+1$ places to insert j into this permutation, exactly two of which are adjacent to i. Each possibility is equally likely. ■

   Rubric: 2 points; this is not the only correct proof.

   (b) Prove that node i is an ancestor of node j with probability $2/(i+1)$. *[Hint: Use part (a)!]*

   Solution: Recall from class that a node x is an ancestor of node y in a priority search tree if and only if, among all nodes with search keys between $key(x)$ and $key(y)$, node x has the smallest priority. Thus, node i is an ancestor of node j if and only if, when we sort the nodes by their search keys, nothing in the set $\{1, 2, \dots, i-1\}$ appears between node i and node j. Equivalently, in the permutation of nodes $\{1, 2, \dots, i, j\}$ induced by the search keys, nodes i and j are adjacent. The result now follows from part (a). ■

   Rubric: 2 points; this is not the only correct proof.

   (c) What is the probability that node i is a *descendant* of node j? *[Hint: Don't use part (a)!]*

   Solution: Zero, because it's a min-heap. ■

   Rubric: 1 point: all or nothing. This *is* the only correct proof!

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

   Solution: The depth of a node is equal to the number of proper ancestors; thus, the expected depth of node j is

   $$\sum_{i=1}^{j-1} \frac{2}{i+1} = \boxed{2H_j - 2} = \Theta(\log j).$$

   ■

   Rubric: 1 point. ½ point for $\Theta(\log j)$. No points for $\Theta(\log n)$.

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

   Solution: The algorithm closely follows the strategy for inserting a node into a treap. First, insert a new vertex with a random search key, using the textbook algorithm for inserting into a binary tree. Then assign this new node the desired priority and rotate it upward to fix the heap property.

   The running time of the algorithm is proportional to the depth of the new node *before* its priority is assigned. If we set the new priority to ∞, this depth is unchanged, but the second phase of the algorithm would do nothing. The new node would have the largest priority in the heater, and so by part (d), its expected depth is $2H_{n+1} - 2 = O(\log n)$. We conclude that the expected running time of our insertion algorithm is $O(\log n)$. ■

Rubric: 2 points = 1 point for algorithm + 1 point for analysis. No analysis credit for just writing $O(\log n)$, or pointing to the lecture notes (which analyze *treaps*, not *heaters*).

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

Solution: We essentially run the insertion algorithm backwards, just as we do for treaps: first rotate the node to be deleted downward until it becomes a leaf, and then discard that leaf.

The running time of this algorithm is proportional to the depth of the leaf just before we discard it. The analysis in part (e) implies that the expected depth of this leaf is $O(\log n)$. ■

Rubric: 2 points = 1 point for algorithm + 1 point for analysis.