Due on Sunday April 11.

1. Let $X$ be a sequence of $m$ accesses to elements in $[n]$. Let $q(i)$, $i \in [n]$ be the number of accesses to elements $i$ in $X$, and let $p(i) = q(i) / m$. Assume $q(i) \geq 1$ for all $i$. Prove that the total search time $\sum_{i=1}^{n} q(i)(d(i) + 1) = m \sum_{i=1}^{n} p(i)(d(i) + 1)$ of $X$ in any binary search tree storing $n$ is $\Omega \left( q(i) \log \left( \frac{m}{q(i)} \right) \right)$. (Hint: One way to do it is using Gibbs inequality which says that $\sum p_i \log \left( \frac{1}{p_i} \right) \leq \sum p_i \log \left( \frac{1}{q_i} \right)$ for any two distributions $p_i$ and $q_i$, $i \in [n]$. Here $0 \cdot \log \frac{0}{0}$ is defined to be 0 for any $x$.)

2. Let $X$ be a sequence of $m$ accesses to elements in $[n]$. Let $q(i)$, $i \in [n]$ be the number of accesses to elements $i$ in $X$. Describe a dynamic programming algorithm that finds an optimal static search tree for $X$. Prove
   1) That your algorithm indeed constructs a tree that minimizes the total access time.
   2) An upper bound on the running time of your algorithm.

3. Give a sequence $X$ for which the algorithm given in class that computes an approximate optimal static tree for $X$ does not compute an optimal tree.

4. Assume we splay at a node $x$. Let $y$ be a node on the path to $x$. Let $d(y)$ be the depth of $y$ before the splay and let $d'(y)$ be the depth of $y$ after the splay. Show that $d'(y) \leq \lfloor d(y)/2 \rfloor + c$ for a constant $c$. What is the smallest $c$ that you can prove this for?

5. We define the following variation on the splay algorithm. This variation looks 3 steps (edges) towards the root from the node $x$ and applies one of the rules in Figure 1 (or their mirror image) if possible. If it is not possible to apply one of the rules in Figure 1 we apply one of the regular zig-zig, zig-zag, or zig rules (Note that zig or zig-zig would apply only if $x$ is at distance 1 or 2 from the root, respectively). Prove that the access lemma holds for this variation as well (with a different constant).

6. Recall the rebalancing operations on 2-4 trees (review this basic material on B-trees if needed).

   When a node $x$ gets too large (has 4 keys and 5 children) then we split it. The parent, $p(x)$, gets an additional key (and child) following the split of $x$ and we split $p(x)$ also if needed. We continue splitting bottom-up until a node does not split or the root splits and we add a new root.

   When a node $x$ looses its last key then its steals a key (and a child) from a sibling if possible and otherwise we fuse $x$ with its sibling. This fusing causes $p(x)$ to lose a key (and a child) and we repeat the process at $p(x)$ if $p(x)$ lost its last key.
1) Describe an implementation of find, insert, delete, join, and split on finger 2-4 trees, that use the rebalancing processes described above. Your implementation should guarantee the following.

Consider a sequence of \( m \) operations on an (initially empty) collection of 2-4 finger search trees. Out of these \( m \) operations \( k \) are concatenations where the \( i \)th concatenation concatenates two trees such that the smaller among them contains \( n_i \) elements. The other \( m - k \) operations are find, insert, delete, and split. The \( j \)-th operation among these \( m - k \) operations is performed on an element of rank \( d_j \) (there are \( d_j \) items smaller than it in this tree) in a tree containing \( n_j \) elements.

Such a sequence should take

\[
O \left( \sum_{i=1}^{k} \log(n_i) + \sum_{j=1}^{m-k} \log(\min\{d_j, n_j - d_j\}) \right)
\]

time. Prove that this indeed holds for your implementation.

2) Improve the bound on the running time of the sequence described above (and improve also your implementation if needed) to

\[
O \left( k + \sum_{j=1}^{m-k} \log(\min\{d_j, n_j - d_j\}) \right)
\].

Figure 1: Splay cases in question (5)