1. Consider two hash tables $T_1$ and $T_2$, both having the same number of slots. With $T_1$, we resolve collisions using chaining, where each chain is a doubly-linked list and insertions are done at the front of the list. With the hash table $T_2$, we resolve collisions using linear probing, and when we delete an element, we replace it with a special marker. We use the same hash function for both the tables.

Let $S : O_1, O_2, \ldots, O_n$ be a sequence of INSERT, DELETE, and SEARCH operations, and suppose we perform $S$ on both $T_1$ and $T_2$. For $1 \leq i \leq n$, let $C_{1,i}$ be the number of element comparisons made when performing $O_i$ on $T_1$, and let $C_{2,i}$ be the number of element comparisons made when performing $O_i$ on $T_2$.

(a) Give an example of a sequence $S$ in which $C_{1,i} > C_{2,i}$ for some $1 \leq i \leq n$, where $O_i$ is a SEARCH operation.

(b) Let $S$ be any sequence of INSERT, DELETE, and SEARCH operations. Suppose we perform $S$ on both $T_1$ and $T_2$, and then perform a SEARCH operation where each item in the table is equally likely to be searched for. Let $Z_1$ be the expected number of comparisons made when performing the SEARCH operation on $T_1$, and let $Z_2$ be the expected number of comparisons made when performing the SEARCH operation on $T_2$. Prove that $Z_1 \leq Z_2$.

2. Draw an example of an AVL tree such that a single removeElement operation could require $\Theta(\log n)$ restructurings (or rotations) from a leaf to the root in order to restore the height-balance property. (Use triangles to represent subtrees that are not affected by this operation).
3. Extend the AVL tree data structure to implement the following method for an ordered dictionary $D$ in $O(\log n)$ time.

\textsc{countInRange}(k_1, k_2):$ compute and return the number of items in $D$ with key $k$ such that $k_1 \leq k \leq k_2$.

Note that this method returns a single integer.

4. Given two strings $a = a_0a_1\ldots a_p$ and $b = b_0b_1\ldots b_q$, where each $a_i$ and each $b_j$ is in some ordered set of characters, we say that a string is \textit{lexicographically less than} string $b$ if either

1. there exists an integer $j$, where $0 \leq j \leq \min(p, q)$, such that $a_i = b_i$ for all $i = 0, 1, \ldots, j - 1$ and $a_j < b_j$, or
2. $p < q$ and $a_i = b_i$ for all $i = 0, 1, \ldots, p$.

For example, if $a$ and $b$ are bit strings, then $10100 < 10110$ by rule 1 (letting $j = 3$) and $10100 < 101000$ by rule 2. This is similar to the ordering used in English-language dictionaries.

The \textit{binary string tree} data structure shown in the figure below stores the bit strings 1011, 10, 011, 100, and 0. When searching for a key $a = a_0a_1\ldots a_p$, we go left at a node of depth $i$ if $a_i = 0$ and right if $a_i = 1$. Let $S$ be a set of distinct binary strings whose lengths sum to $n$. Show how to use a binary string tree to sort $S$ lexicographically in $\Theta(n)$ time. For the example in the figure, the output of the sort should be the sequence 0, 011, 10, 100, 1011.

![Figure 1: A binary string tree storing the bit strings 1011, 10, 011, 100, 0. Each node’s key can be determined by traversing the path from the root to that node. One can store the key at the node. Nodes are heavily shaded if the keys corresponding to them are not in the tree; such nodes are present only to establish a path to other nodes.](image-url)