Homework Assignment #6
Due: November 1, 2024 at 5:00 p.m.

1. Consider a set ADT that only has to support DELETE and FIND operations (no INSERTS).

Suppose we use an external (leaf-oriented) BST to represent the set. Initially, the tree contains n_0 keys of the set and is perfectly balanced so that its height is exactly $\lceil \log_2 n_0 \rceil$. The value of n_0 is known. The argument to a DELETE operation is a pointer to the leaf that must be deleted.

Our goal is to design a data structure so that in any sequence of FIND and DELETE operations, the *worst-case* time for a FIND is $O(\log n)$ (where n is the number of keys stored in the set *at the time* the FIND is performed) and the *amortized* time for each DELETE is $O(1)$.

The idea is that a DELETE does not actually remove the leaf. Instead, it simply marks the leaf (by setting a bit stored in the leaf), to indicate that the leaf's key has been deleted. Then, periodically, the entire tree is rebuilt to again make it a perfectly balanced BST containing all the unmarked leaves.

[6] **(a)** Describe how to rebuild the tree efficiently. If the old tree contains n leaves, how much time does it take to do the rebuild? Give your answer in terms of n using Θ notation and *briefly* justify your answer.

[1] **(b)** Explain how would you decide *when* to rebuild the tree in order to achieve the time bounds described above.

[3] **(c)** Show that the amortized time per DELETE is $O(1)$.

[2] **(d)** Show that the worst-case time per FIND is $O(\log n)$ when the tree has n (unmarked) keys.