Local Minima in Binary Trees:

Problem:

Consider an n-node binary tree T, where $n = 2d - 1$ for some d. Each node v of T is labeled with a real number X_v . Assume that the real numbers labeling the nodes are distinct. A node v of T is a local minimum if the label X_v is less than the label X_w for all nodes w that are joined to v by an edge. Find a local minimum of T evaluating only $O(logn)$ nodes of T.

Solution:

The goal in this problem is to find the local minimum in a given tree, T, by evaluating only $O(logn)$ nodes of T. There are two possible candidates for a local minimum in T: a leaf node and a root node. For a leaf to be a local minimum, it must be smaller than its parent node. A root node can be a local minimum as long as it is smaller than **both** of its children.

In order for this algorithm to work, we need to check, at most, two nodes at each level of the tree until we find the local minimum. In other words, if we have an algorithm that takes d steps and it takes only a constant number of operations per step, then the algorithm for finding the local minimum in T will run in $O(logn)$ time.

The algorithm for finding the local minimum in T runs as follows:

1. Begin at the root node of T. This will be the 'current node' in our first comparison.
2. Check to see if the current node is less than both of its children.
3. If the current node is less than both of its chidlren, it is the local minimum. If not, continue the search, since at least one child must have a value that is less than the root.
4. At each successive level of T, repeat steps 2 & 3 until we find the local minimum.

The above algorithm takes only a constant number of steps, since it is only doing a constant number of comparisons at each node-level. In other words, in order to find the local minimum of T, the algorithm only needs the values of **at most** two nodes at each level.

Expressed in a more formal way, we can claim that with d levels, we are checking $2(d-1)$ nodes. Alternatively, we can claim that we are making a total of $2(d-1)$ node comparisons. The reason we are only conducting $2(d-1)$ evaluations is that we are making 2 comparisons per level in T, except at the root of T, where we only make 1 comparison.

This algorithm is clearly $O(logn)$. At each node-level, the algorithm splits T (by choosing the smaller node) and executes a constant number of operations.