Adrian deCola
Data Structures and Algorithms

Homework 6- Written Exercises

1. What does it mean if a binary search tree is a balanced tree?

For a binary search tree to be balanced means that for any node, the height of its left subtree and right subtree do not vary by more than 1. This also means that we can search for elements in O(log(n)) time.

2. What is the big-Oh search time for a balanced binary tree? Give a logical argument for your response. You may assume that the binary tree is perfectly balanced and full.

The search time for a balanced binary tree is O log(n). To show this, let's assume a perfectly balanced and full binary search tree. This means that each guess we remove at least half of the nodes. In the worst-case, we much do this down the height of the tree. Therefore, the search time is the height of the tree. Considering the worst case, we have eliminated at least half the nodes each time we traverse until we are left with one node. This means that $1 = n\left(\frac{1}{2}\right)^h$, where n is the number of nodes and h is the height of the tree. Solving this we find: $n = 2^h$ or $h = \log(n)$.This means the search time, which is the height of the tree, is O(log(n)).

3. Now think about a binary search tree in general, and that it is basically a linked list with up to two paths from each node. Could a binary tree ever exhibit an O(n) worst-case search time? Explain why or why not. It may be helpful to think of an example of operations that could exhibit worst-case behavior if you believe it is so.

Yes, a binary tree could exhibit a O(n) worse-case search time. One example this is if all the nodes only have a right child or no child (for the only leaf node). This binary search tree would be a long tree extending down the right and if we wanted to search for the largest node the search time would be O(n). This is why it is important to balance out binary search tree if we want to be able to search it in O(log(n)) time.

4. What is the recurrence relation for a binary search? Your answer should be in the form of T(n) = aT(n/b) + f(n). Clearly state the values for a, b and f(n).

The recurrence relation for a binary tree is: $T(n) = T(\frac{n}{2}) + 1$, as each time we eliminate half the list and must do one comparison.

5. Solve the recurrence for binary search algorithm using the substitution method. For full credit, show your work.

Using the substitution method, we find a pattern:

$$T(n) = T(\frac{n}{2}) + 1$$

$$T(n) = T\left(\frac{n}{4}\right) + 2$$

$$T(n) = T(\frac{n}{8}) + 3$$

…

$$T(n) = T(\frac{n}{2^k}) + k$$

Assuming n is of the form: $n = 2^k$.

$$T(n) = T(1) + k$$

We know for a tree of size one we must only do one comparison. Substituting in k:

$$T(n) = \log(n) + 1$$

This means that the search time is on the order of O(log(n)).

6. Confirm that your solution to #5 is correct by solving the recurrence for binary search using the master theorem. For full credit, clearly define the values of a, b, and d.

We can also show this using the master theorem. The master theorem requires the recurrence relation to be in the form:

$$T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$$

Our equation is in this form where $a = 1$, $b = 2$, and $d = 0$. $log_b(a) = d = 0$. Therefore, the master theorem states:

$$T(n) = O(n^d \log(n)) = O(\log(n))$$