Quiz 6 Feedback CS2852, Dr. Yoder, Spring 2014

Problem 1 – Computing the product of node values recursively

Choosing the recursive call

① -0.5 node.value is included twice in the final product, resulting in its value being squared. You can avoid this by not passing the product on to the recursive calls… and just accepting their product back to be included as one term in the product, e.g. for a full node, `prod(node.right) * node.value * prod(node.left)` (Though in practice you would not compute the product this way since node.right might be null.)

② -0.5 Only one of the children can possibly be included in the sum at any given time. This is why we avoid early-returns except for parameter checking. (And in some cases, some of the code is unreachable because the code will ALWAYS return before reaching a given line.) (See also ⑧.)

③ -0.5 Local variable product is not updated during recursive call. Don't make an instance variable to fix this, but rather make use of the `int` product returned by the recursive call. (See also ⑧.)

⑬ -0.5 Must check that node.left and node.right are not null.

⑮ -0 Should compute product, not sum.

Base case

④ -0.5 Need to specify what "value" should be for an empty sub-tree.

⑭ -0.5 Value is omitted from product – as a result we will never actually add anything to the product. If it starts at 1, it will remain 1.

⑤ -0 default value for prod is 1, not 0.

Root of tree

⑥ -0.5 Need to start from the root of tree, which requires writing a private helper method.

⑦ -0 Need to check that root is not null before calling the private helper method.

Wrap-up

⑧ -0.5 Error during wrap-up. This can be avoided by maintaining an `int result` and only `return`ing it at the end. Throughout the method, the result can be `*=` by each new thing to be included in the product. (See also ②.)

Recursive call

⑨ -0.5 Must specify somehow that the recursive calls should go left or right down the tree, e.g. make private helper method accepting a node that can be called like `prod(node.left)`.

Minor remarks

⑩ -0 Simplify – if(a && b) {..} else if (a) {…} else if (b) {…} else {…} can often be simplified to
.. if(a) {..} if (b) {…} … with appropriate changes in the …'s.

⑪ -0 must return from method

⑫ -0 cannot call recursively on the node like `current.left.prod()` because left is a `Node`, not a `BinarySearchTree`. Instead, need to create private helper method that can be called as `prod(current.left)`.

⑯ -0 Don't need to call accessor methods – Node is private inner class and all its instance variables can be accessed from anywhere.