Algorithm:

1. If the nodes in comparison are both null then they can be deemed symmetric.
2. If the nodes in comparison are both not null; and that their values match, then recursively check the symmetry of first node's left subtree and second node's right subtree, and if that is true, recursively check the symmetry of first node's right subtree and second node's left subtree.
3. If either of the nodes in comparison are null and the other is not, then they can not be symmetric so return false.

Test:

1. Null as the root node.
2. Left/right skewed tree.
3. Tree with identical left and right subtree nodes.
4. Tree with mirror left and right subtree nodes.

Solution:

```
/**
     * Definition for a binary tree node.
     * public class TreeNode {
     *     int val;
     *     TreeNode left;
     *     TreeNode right;
     *     TreeNode(int x) { val = x; }
     * }
     */
    class Solution {
        public boolean isSymmetric(TreeNode root) {

            //call the recurisve helper method passing root, root as the root of left and right subtrees
    initially
            return helper(root, root);

        }

        //helper method that gets t1 and t2 as root of left and right subtrees
        private boolean helper(TreeNode t1, TreeNode t2) {
            //if t1 and t2 are both null then return true
            if (t1 == null && t2 == null)
                return true;

            //if either t1 or t2 is null then return false
            if (t1 != null && t2 != null)
            //return t1.val == t2.val && helper(t1.left, t2.right) && helper(t1.right, t2.left)
                return (t1.val == t2.val)  && helper(t1.left, t2.right) && helper(t1.right, t2.left);

            return false;
        // T O(n) S O(n) due to recursion
        }
    }
```