Part 1:
7.)

    height(): O(n)
    parent(): O(n)
    level(): O(n)
    isComplete(): O(n)
    isPerfect(): O(n)
    hasDoubles(): O(n)

8.) To prove that in a complete tree T, the index of every node is less than n, where n is the number of nodes in T, we will be using standard mathematical induction.

Claim: In a complete tree T, the index of every node is less than n, where n is the number of nodes in T.

Base Case: Let the number of nodes in a tree be represented by n and index represented by i

Now consider a complete tree with n = 1 and no leaves. In this case, the single node is at level 0. In this case, since the root is the only node in the tree, we know that i = 0.
The index to the left of the node is defined as 2^y - 1 where y is the level of the leaf so:
2^0 - 1 = 0
The index to the right of the node is defined as 2^y+1 - 2 where y is the level of the leaf so:
2^(0+1) - 2 = 0

Thus, since the left and right index of a complete tree with n = 1 nodes is 0, the claim holds true for this base case and n > i.

Inductive Hypothesis: Assume that the above claim holds for all nodes k such that k > 1 and that the indices of the left and right most leaves are 2^y - 1 and 2^(y+1) - 2 respectively.
Inductive Step: Let k >= 1, Assuming that nodes of value k hold, we can prove that the claim holds for k + 1 nodes

Now let us consider a complete tree with k + 1 nodes, since we're increasing the number of nodes from the original complete tree T, this means that the total number of nodes has now increased. Since the new node that was added is just (index of last node) + 1 since the claim holds for k we can assume it holds for k + 1 since both the total number of nodes and indices increases by 1.

1) From our inductive hypothesis, the index of the left-most leaf in a tree with nodes k is (2^y) - 1, where y is the level of the leaf, which is less than the total number of nodes k. For a tree with k + 1 nodes we do (2^y) > k + 1 by adding + 1 to each side of the

previous expression with k nodes. Since 1 is added to each side the expression on the right is still greater than the expression on the left and so our claim that the number of the indices is less than the total number of nodes k + 1 is true.

2) From our inductive hypothesis, the index of the right-most leaf in a tree with nodes k is (2^(y+1)) - 2, where y is the level of the leaf, which is less than the total number of nodes k. For a tree with k + 1 nodes we do (2^(y+1)) - 1 > k + 1 by adding + 1 to each side of the previous expression with k nodes. Since 1 is added to each side the expression on the right is still greater than the expression on the left and so our claim that the number of the indices is less than the total number of nodes k + 1 is true.

*Therefore, we have proved that the indices in a complete tree are less than k + 1 nodes.

9) A bound function for the recursive method recIsComplete() is f(r) = number of nodes a complete tree can have for root r. Correct Bound Function: f(r) = 1.

1) f(null) = 1
2) f(r) > 1 when r > 1
3) f(r.left) < f(r) and f(r.right) < f(r)

Consider a proof on the correctness of the method recIsComplete(root, index). A proof by structural induction will be conducted.

Claim: recIsComplete(root, index) returns true if the root of the tree with root *root* is complete

Base case:

If r = null, recIsComplete(root, index) returns true

Inductive Hypothesis:
Assume f(k) is a complete tree where k is the number of nodes in a tree

Inductive claim: f(r) holds for any subtree of T where r is the parent of c

Inductive step:
If the parent is null, then the child c is considered as the root and no other nodes are in the tree so this would result in a complete tree.

From the inductive hypothesis, if the parent is not null, then the function correctly calculates recIsComplete(c, index) for the

There are 4 cases we have to consider: If c has both left and right children, then c is only complete if both c.left and c.right are complete. If c has only 1 left child and 0 right children then c can only be complete if it's the last node on the current level and c.left is complete. If c has only a right child it can’t be complete as the left child is missing. If c has no children and c is a leaf node it is complete. The code correctly implements this logic.

Therefore we have proved the claim that recIsComplete(root, index) correctly calculates whether or not a tree is complete.