Questions

1. Done

2.
    a.
        i. Node a: level 3
        ii. Node f: level 1
        iii. Node g: level 4
        iv. Node k: level 0
        v. Node l: level 2
        vi. Node p: level 2

    b. Height of the tree is 4 because the nodes farthest away from the root are at level 4.

3. (see code)

4. (see code)

5. Tree A is unbalanced because not all nodes exist at the height (4) or at height - 1. In other words, all nodes do not exist within one level of each other. Node with 3 is at level 2 whereas several nodes exist at level 4.

   Tree B is balanced because all nodes are either at the level of the height (4) or at level 3. So, all nodes exist within one level of each other.

6. (see code)

7. Tree C is height-unbalanced because right away from the root, the left subtree has a height of 3, while the right subtree has a height of 1. This is already greater than the difference of 1 level that is allowed for it to be considered height balanced.

   Tree D is height-balanced because for every node in the tree, the left and right subtrees have a height of within 1 level of each other.

8. (see code)

9. i and l are the only options because they are the in-order predecessors and successors respectively (i.e. in ascending order, i is the element that is immediately before k, and l is the element that is immediately after k). If any other value is moved to the root, the

ordering of the tree will be lost. The correct ordering of a tree is when everything to the right comes after the node and everything to the left comes before, for a respective comparison.

10. (see code)

11. (see code)

12. The way isLongest() works is that we do a comparison at each node between all the nodes connecting to that parent node and the node we're looking it. The method, isLongest() only makes one iteration through the tree of n nodes, so we are only looking at each node once. Then the next thing to look at is the work done at each node. For work done at each node, we have three different possibilities. One is where the node is a leaf (no children), one is where the node has only one child, a right child or a left child, and one is where the node has two children, both a left child and right child. For when a node is leaf, we have a simply return of Entry at that node, which is simple return operation of o(1). For when a node has only one child, we have a comparison between the two Entry nodes and return of the Entry which has longest firstName field. Therefore we a comparison and a return; since both are fixed length tasks, we get o(1) for this case as well. For when a node has two child nodes, we have two comparisons and a return of the desired Entry. Since we have to decide which Entry has longest firstName, we do 2 comparisons to find from 3 Entry objects. Again, comparisons and the return are fixed, so we get o(1) again. None of these depend on how many nodes or height that tree has, but instead a fixed task done for each node. Therefore looking at the overall method, we get **O(n)** where n is the number of nodes of the tree.

Testing of Methods

Note: All methods tested in the main method

Method	Test Case	Parameters (if any)	Expected Output	Actual Output
nodeLevel	Data does not exist in the tree	data	-1	-1
	Data exists in a node (not root) in the tree at level 2	data (`stu2`)	2	2
	Data is at root	`stu1`	0	0
height	Empty tree	`tree` with nodes initialized	-1	-1
	Tree with only one node: root, and no subtrees	`tree` with only stu1 node	0	0
	Tree with multiple nodes from lab	`tree` with student and faculty data from lab inserted	4	4
isBalanced	Empty tree	`tree` with nodes initialized	True	True
	Tree with only one node: root, and no subtrees	`tree` with only stu1 node	True	True
	Tree with node which has left data but not right data	`tree` with aforementioned data inserted	True	True
	Tree with node which has right data but not left data	`tree` with aforementioned data inserted	True	True
	Tree with multiple nodes from lab	`tree` with student and	True	True

		faculty data from lab inserted		
	Tree with student and faculty data from old lab inserted, with one node on right of root (making tree unbalanced)	`tree` with aforementioned data inserted	False	False
isHeightBalanced	Empty tree	`tree` with no nodes initialized	True	True
	Tree with only one node: root, and no subtrees	`tree` with only stu1 node	True	True
	Tree with node which has left data but no right data	`tree` with aforementioned data inserted	True	True
	Tree with node which has right data but not left data	`tree` with aforementioned data inserted	True	True
	Tree with multiple nodes from lab	`tree` with student and faculty data from lab inserted	False	False
	Tree with student and faculty data from old lab inserted, with one node on right of root (making tree unbalanced)	`tree` with aforementioned data inserted	False	False
remove	Empty tree	`tree` with no nodes initialized	Exception thrown	Exception thrown
	Removing root from tree with data from lab	`stu1`	“TheTerry Walker” is removed and “John	“TheTerry Walker” is removed and “John Stone”

			Stone" becomes new root	becomes new root
	Removing non-root node from tree with data from lab; node has no right child	`stu4`	Node is removed and left child (Muffin the cat) is promoted	Node is removed and left child (Muffin the cat) is promoted
	Removing non-root node from tree with data from lab; node has no left child	`stu2`	Node is removed and right child (Donna Marie) is promoted	Node is removed and right child (Donna Marie) is promoted
	Removing non-root node from tree with data from lab; node has both left and right children	`stu3`	Node is removed and the node that has firstName that is alphabetic-ally after removed node is promoted	Node is removed and the node that has firstName that is alphabetically after removed node is promoted
	Removing non-root node from tree with data from lab; node is a leaf	`fac4`	Node is removed and no node is promoted	Node is removed and no node is promoted
findLongest	Empty tree	`tree` with no nodes initialized	null	null
	Tree with only one node: root, and no subtrees	`tree` with only stu1 node	Entry for TheTerry Walker	Entry for TheTerry Walker

	Tree with multiple nodes and only one node has max firstName length	`tree` with aforementioned data inserted (TheTerry => Terry)	Entry for Shamrock The Cat	Entry for Shamrock The Cat
	Tree with multiple nodes and left subtree has node with max firstName length	`tree` with aforementioned data inserted	Entry for “TheTerry Walker” and not “Shamrock The Cat”	Entry with max firstname length for which the first name comes last in dictionary order

Since our actual testing outcomes are consistent with our expected outcomes, we can surmise that all methods work as intended.