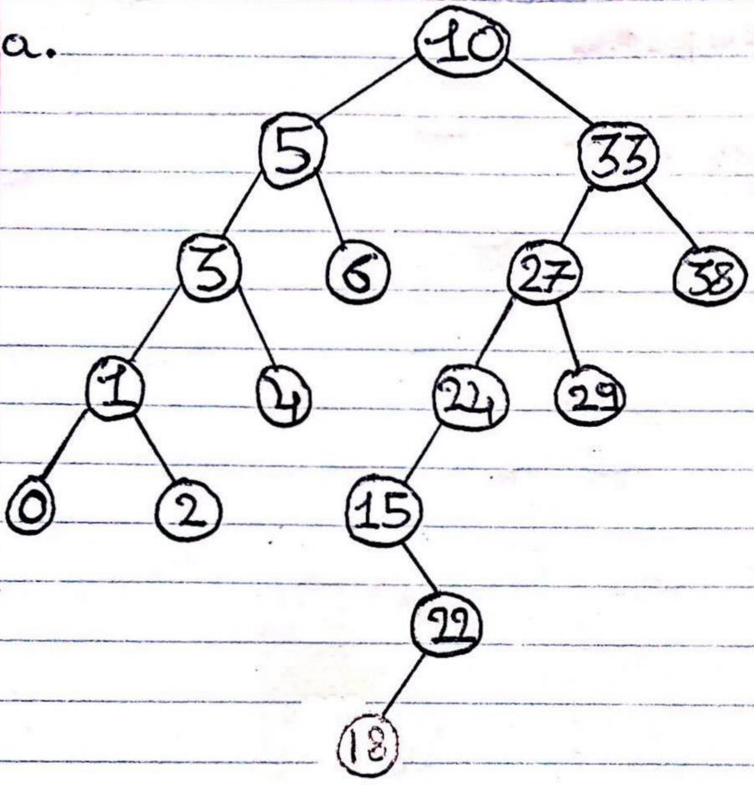
CSI2110-D – Written Assignment #5

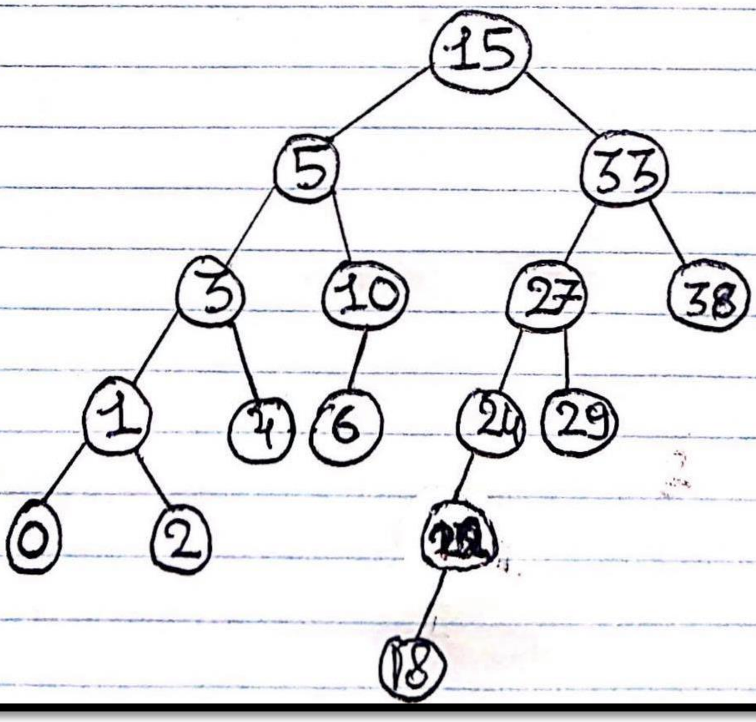
Alae Boufarrachene (300188539)

Question 1 :

a.



b.



c)

-inOrder() Traversal of **a)** : 0,1,2,3,4,5,6,10,15,18,22,24,27,29,33,38

-inOrder() Traversal of **b)** : 0,1,2,3,4,5,6,10,15,18,22,24,27,29,33,38

The outputs are the same. This is due to the inherent structure of a binary search tree and how the left descendants (left sub-tree) of a given node are only composed of inferior elements, and the right descendants (right sub-tree) of said node are only composed of superior elements.

Meaning that the tree would be printed in ascending order when performing an inOrder traversal.

Moreover, in both the tree removals we performed in (a) and (b) we deleted the same element (the root), the remaining elements are the same, which results in the same inOrder traversal output for both trees.

<u>Question 2 :</u>

a)
-Function layout *: thirdLargestElementBST(node)*

-Input : Root of given tree

-Algorithm (assuming the input are well-entered and don't result in edge cases) :
➥**Step 1 :** We declare a counter variable that's initially equal to zero (it'll be used to keep track of the amount of nodes traversed).
➥**Step 2 :** We perform a reverse inOrder traversal of the tree (following the Right-Root-Left pattern) which would be done at each node recursively.
➥**Step 3 :** We increment the counter each time we move on to the next node.
➥**Step 4 :** We stop making recursive node calls when the counter becomes equal to 3 and store the data of the current node in a variable (by using a conditional statement that checks counter
➥**Step 5 :** We return the variable that stores the data of the 3rd largest element.

b)
My solution goes all the way down to the largest element of the tree (which is the deepest rightmost node in the case of a BST) and traverses 3 elements, which makes my solution O(h+3), which can be simplified to O(h).

c)