Ata Seren 21901575 CS202-1

Answer for Question 1:

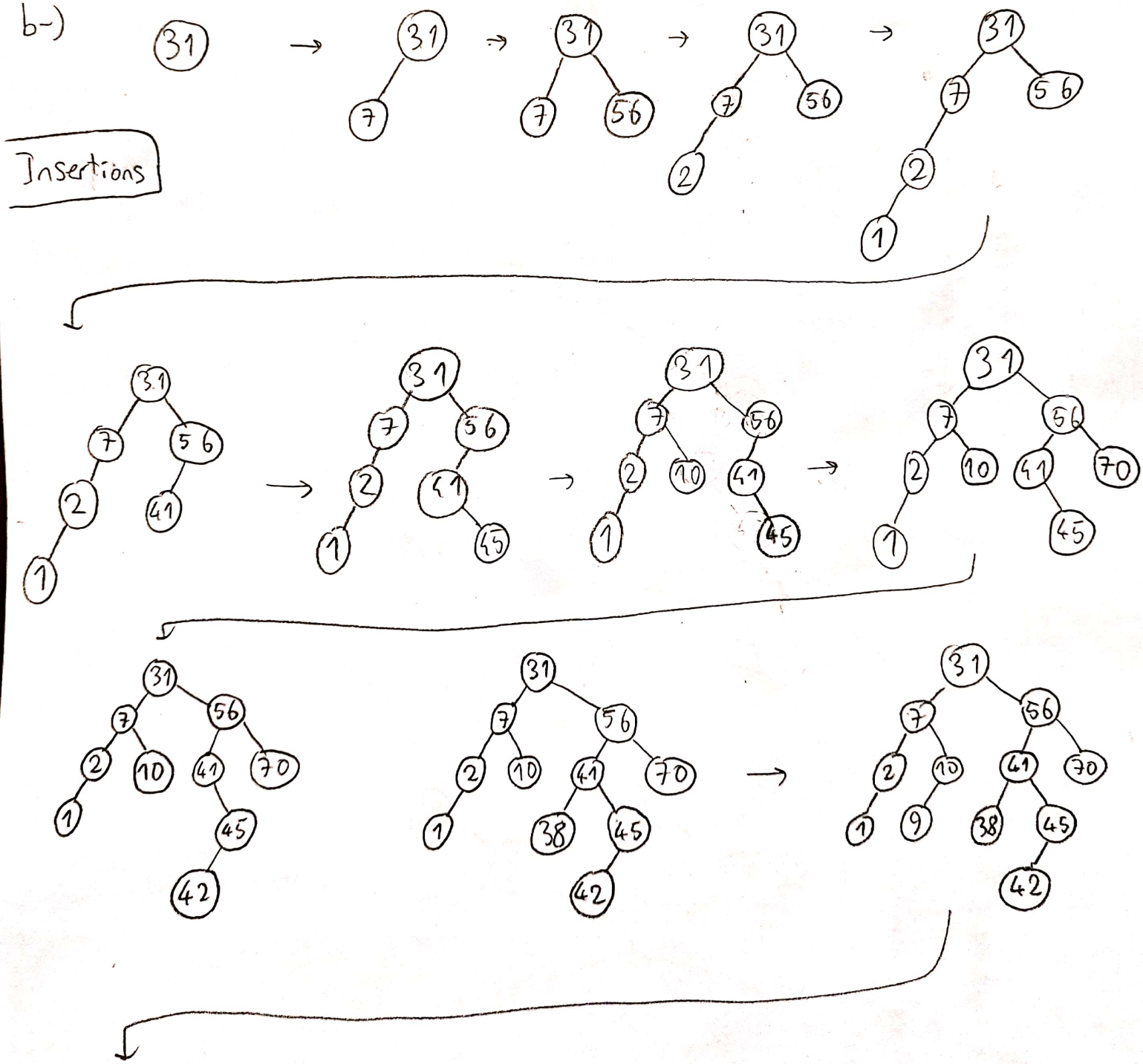
a-)

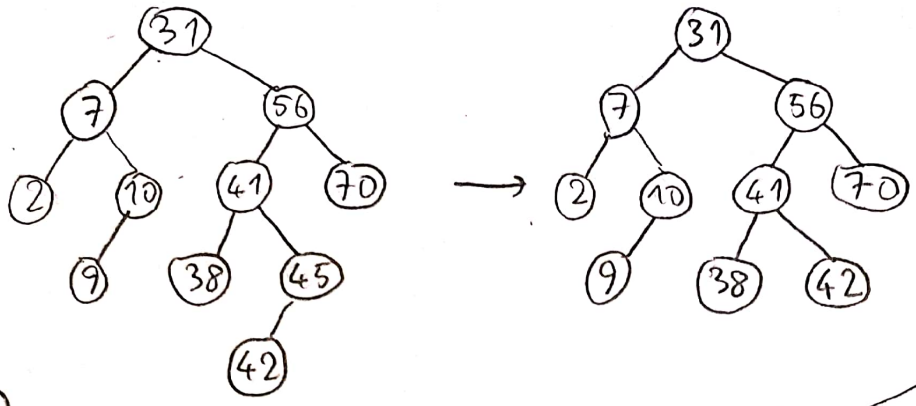
Prefix expression: /*A+BCD

Infix expression: A*B+C/D

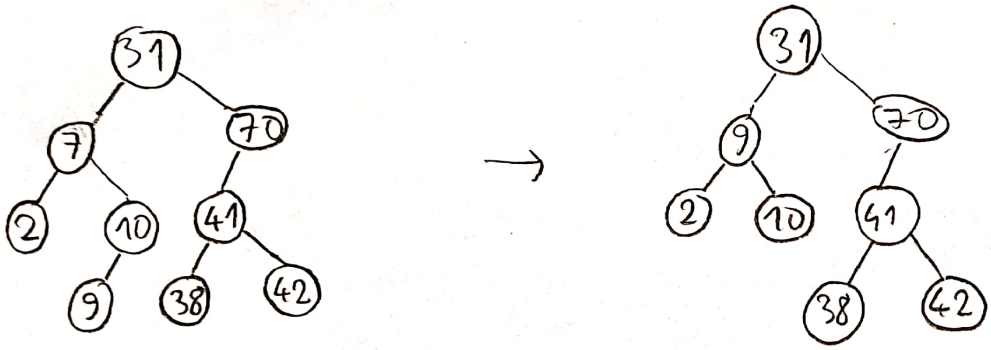
Postfix expression: ABC+*D/

b-)

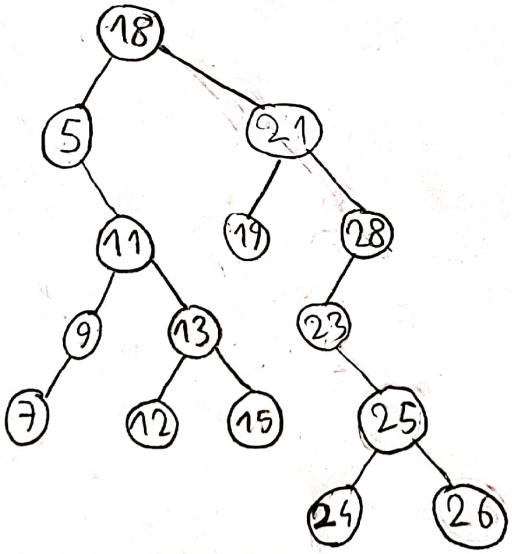




Deletions



(-) Corresponding BST:



Postorder traversal: 7, 9, 12, 15, 13, 11, 5, 19, 24, 26, 25, 23, 28, 21, 18

Answers for Question 3:

levelorderTraverse: In levelorderTraverse() method, I implemented a for loop that gets height of tree as a limit. I got height using a helper method I used for getHeight() method. In this for loop, I used a helper method called printLevel that gets a node and integer for level of tree. This printLevel method gets a value for level as parameters. Then, it recursively reaches to the nodes at that level, by decrementing level value in each recursion. When it reached nodes in desired level, it prints each node in called recursive functions. Function prints values from left to right because in every recursive call, it first reaches nodes in left, then right.
Its worst case complexity is O(n^2). First of all, there is a for loop in levelorderTraverse method which gives us O(n). Then, helper method printLevel calls itself until level value becomes 1, which also gives us O(n). By multiplying these complexities (because they are nested), we get O(n^2). Time depends on height of the tree. So, tree which each node has only one child, results in maximum height and therefore, worst case.
LevelorderTraverse could be implemented by using a queue and its complexity would be O(n) because a queue could eliminate the need for a for loop and decrease the running time.

span: In span() method, I implemented a helper method called spanHelper which takes a node, interval values and an additional count integer to count number of nodes in the interval as parameters. In spanHelper method, if item of given node is bigger than small value of internal, it calls itself and goes with left side of the node. Then, if the item is in interval, it increments count integer as 1. Also, it calls itself and goes with the right side of the node, if it is smaller than big value of interval. Finally, after recursion ends, span method returns count value.
Its worst case complexity is O(n) because in worst case which every item is in interval, it should visit every node.
I don't have any idea to implement it in a faster way.

mirror: First of all, I implemented a helper method for recursion called mirrorHelper and it takes a node as parameter. This mirrorHelper method first calls itself and takes left node of first node as parameter. In the same way, mirrorHelper is called again and it takes right node of first node as parameter. At the end, left node of first node is saved temporarily, right node initialized to left node and temporary node initialized to left node. In this way, left and right nodes are swapped.
Its worst case complexity is O(n) because it always has to traverse all of the tree and because of that, it can't be implemented in a faster way.