CS 202

Fundamental Structures of Computer Science

Homework 2

Berdan Akyürek

21600904

Section 2

1)

a)

Preorder (Prefix) (Root-Left-Right):

/ * A + B C D

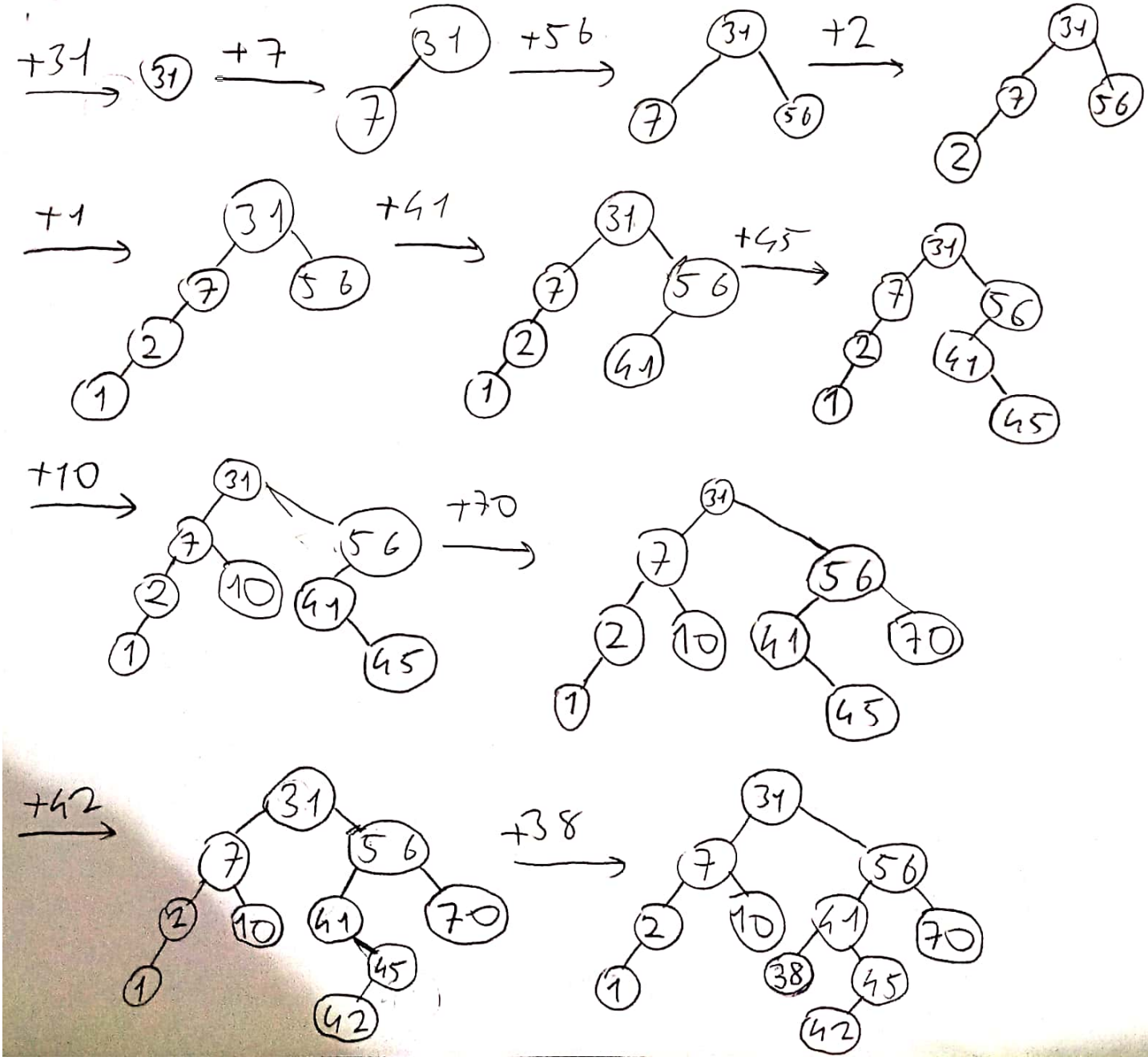
Inorder (Infix) (Left-Root-Right):

A * B + C / D

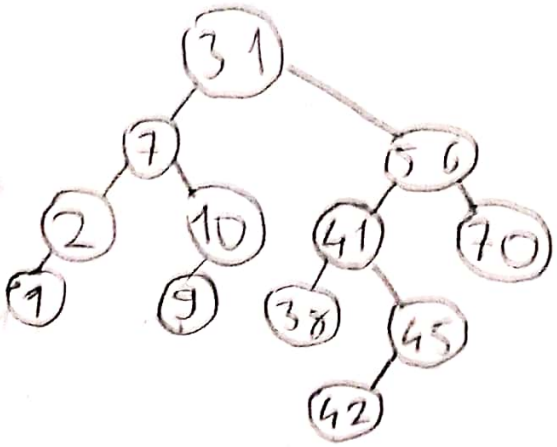
Postorder (Postfix) (Left-Right-Root):

A B C + * D /

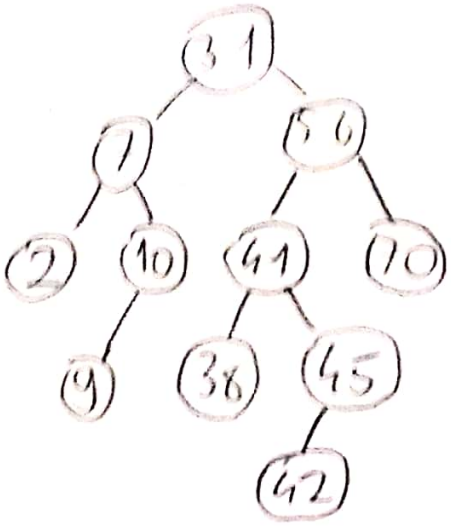
b)



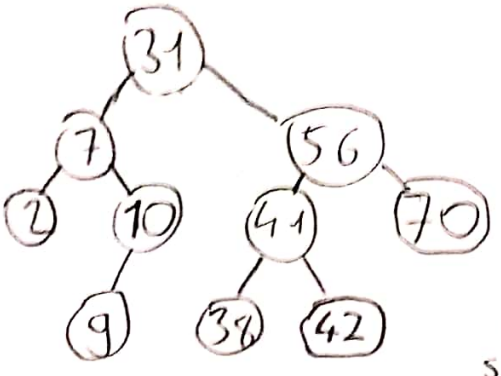
+9



-1

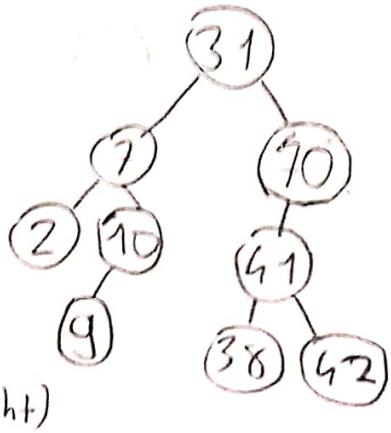


-45



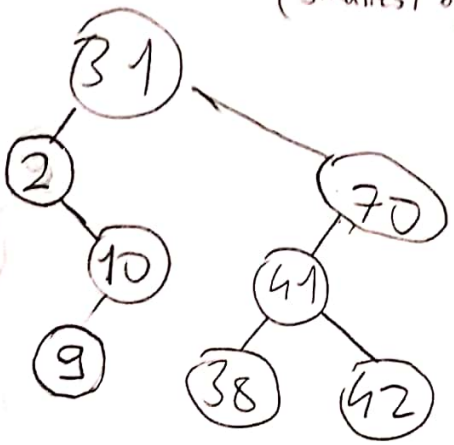
-56

replacing with inorder successor (70) (Smallest of right)

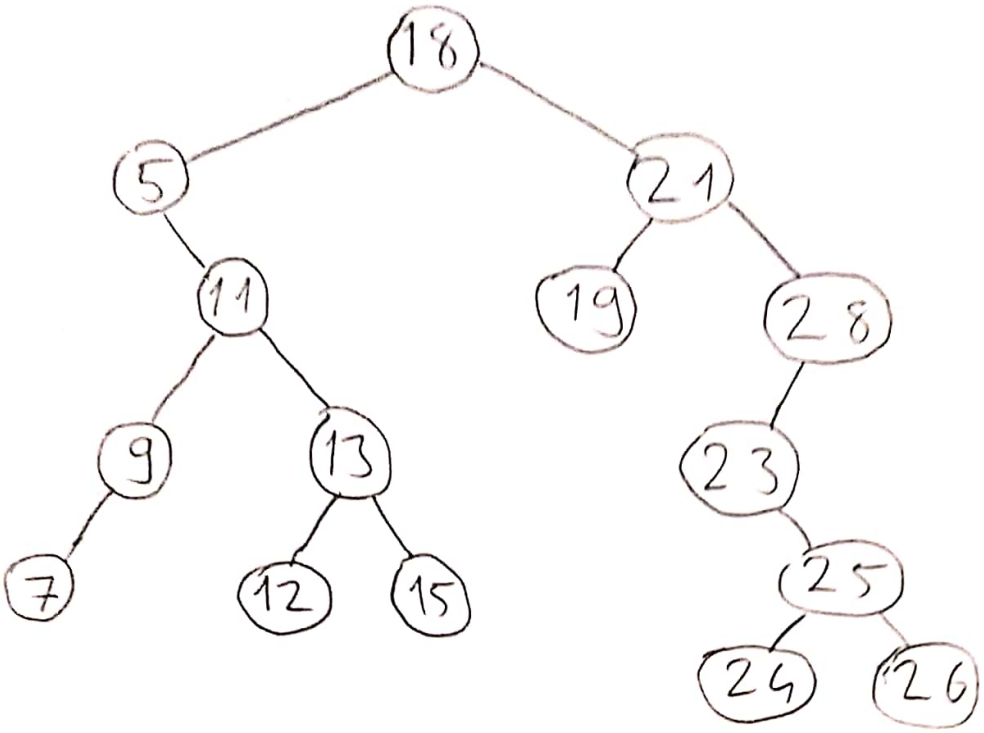


-7

replacing with inorder Predecessor (2) (Largest of left)



c) The BST with given preorder traversal:



Its postorder traversal (Left-Right-Root):

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

3)

Analysis of the three given functions are as follows:

levelorderTraverse():

I implemented this function using queue data structure. Function starts by creating an empty queue and inserting the root node into it. Later, until the queue is empty again, it takes an element from the queue, prints it and inserts its left and right childs back to the queue (if they exist). This way, each level is executed in order, from left to right.

The body of the loop is executed n times, where n is the number of nodes in the three. So the running time complexity is O(n).

Since every node in the tree has to be printed, every single node has to be visited. So to find a better solution than O(n) is not possible for this function.

span():

I implemented this function recursively. Starting from the root node, the program checks if the current node is in the interval and calculates the result of left and right subtree recursively. But when it is certain that a subtree cannot contain any value in the given interval, that subtree is not examined to increase efficiency. Because this is a BST and it is certain that the left subtree is always smaller compared to the parent and the right subtree is greater. If the parent is outside of interval or right in the edge, only one of the two subtrees can contain nodes in the given interval. So visiting others is not necessary.

In the worst case, where every single node is in the interval, the function is called once for each node in the tree. So the worst case complexity is O(n).

For the worst case, every node has to be visited to make sure to count them. So in the worst case, it is not possible to find a solution better than O(n).

Also in the best case, there could be only a single subtree and root nodes that have a single child and this subtree can be completely out of bound. In this case, the best case is O(1). It is not possible to find a better solution in the best case too.

But this function uses additional space and it is possible to use less space and reduce space complexity.

mirror():

I implemented this function recursively too. It swaps the left and right childs and does the same for each child until leaf nodes are reached.

Again, since it is executed for each node, complexity is O(n) where n is the number of nodes in the tree.

Since, to make swap operations, every node needs to be visited. This shows it is not possible to do this operation with a time complexity better than O(n).