Ahmed Salih Cezayir/21802918/Section2

CS202- Assignment 2: Binary Search Trees

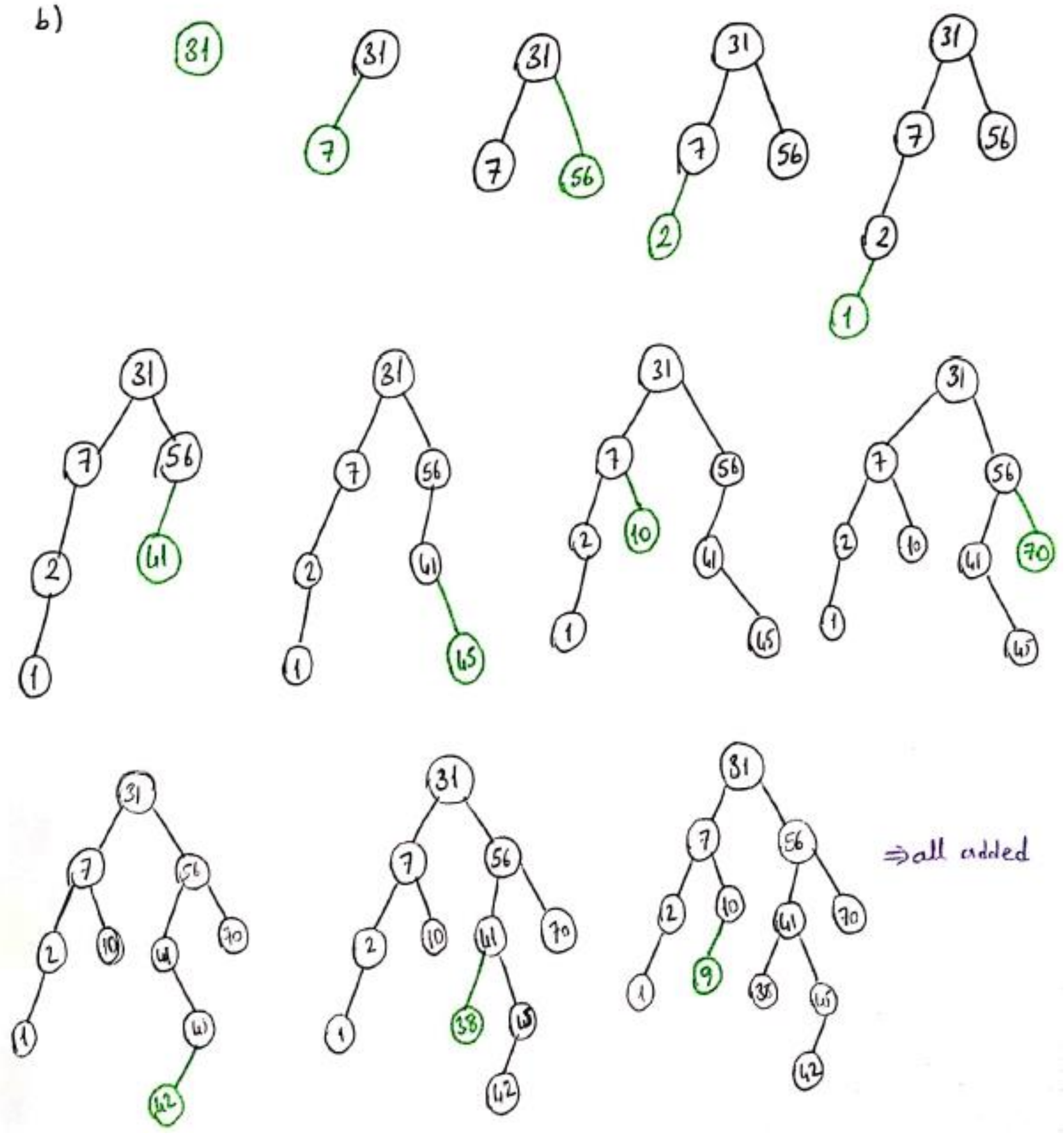
Question 1

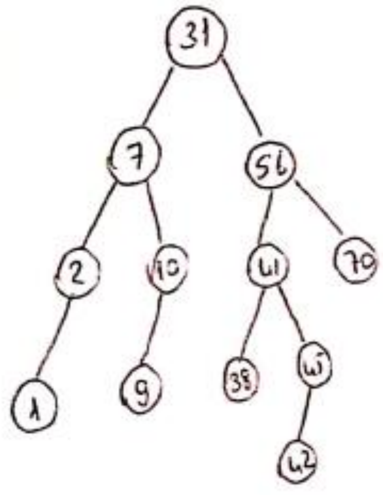
a) prefix : / * A + BCD

infix : [A * (B + C)] / D

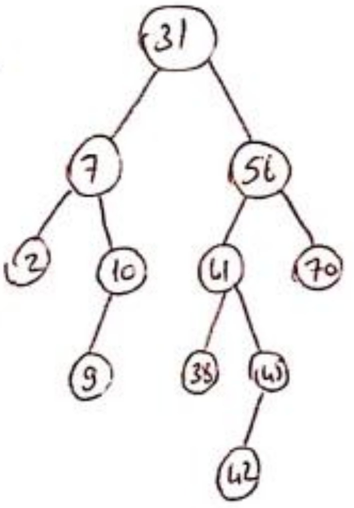
postfix : ABC + * D /

b)

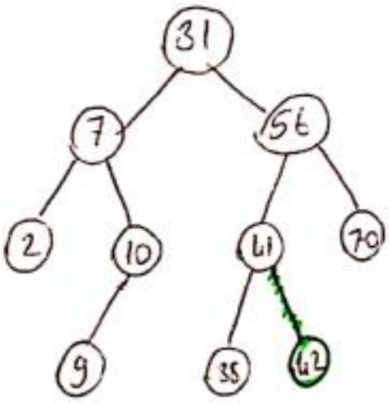




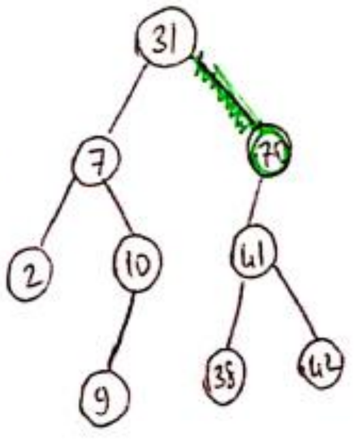
delete 1 →



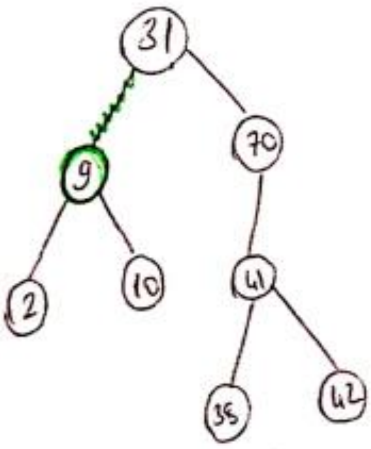
↓ delete 45



↓ delete 56

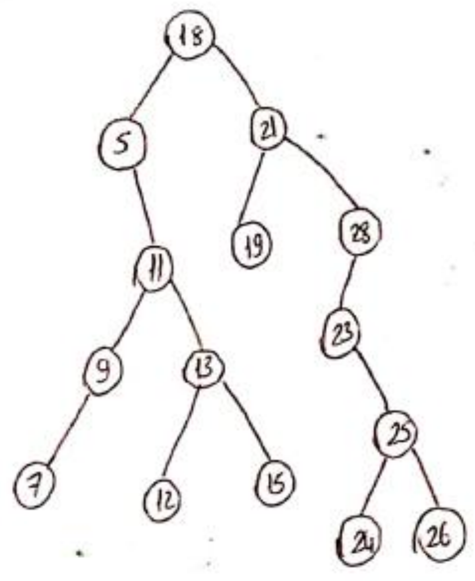


delete 7 ←



(c)

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



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

Question 3:

In all the three functions (levelorderTraverse, span, and mirror), I used helper functions to use the root and recursion.

LevelorderTraverse

To implement the levelorderTraverse function in the helper function, I used a for loop and another helper function in the loop to print the nodes in one level. I looped 'height' times because there were maximum 'height' levels. In each iteration, in the helper function to print the nodes in one level, I first checked which level we are in, and if we are in level 1, I printed the item in the root. If the level is bigger than 1, by using recursion, I called the same method for right and left subtrees with 'level - 1' level. In this way, I could print each level. For instance, to print the second level, I first call the helper function with level = 2. Since the level does not equal to 1, I just recursively called this method again for the right and left subtree with level = 1. In each recursion, since we also go into a recursion for both left and right subtrees, we can print all the same level nodes.

Worst Case Complexity: If the BST is a long one with only right or left children, we get maximum height which is n. Because of this, the outer loop will be executed n times. In the first iteration, we will just print the root, so it has O(1) complexity. But with each iteration, the recursion amount will increase. $T(n) = O(1) + ... + O(n-1) + O(n) = O(n^2)$. So, the worst-case complexity is $O(n^2)$. Since the worst case is just a linked list, we can traverse it in O(n), so maybe we can make some adjustments and decrease the worst-case time complexity to O(n).

Span

To implement the span function, in the helper function, I first check the root pointer, and if it is 0, I return 0. After that, I check the root whether it is in the range. If it is in the range, I call the same method for right and left subtrees and return the sum of them + 1 (This plus one comes from the root). If this condition is not correct, then the root must be out of boundaries. If it is bigger than the range, I call the method for the left subtree(because the left subtree has the smaller values). Likewise, I do the same thing for the situation where the root is smaller than the range. In that case, I call the method for the right subtree. In this way, I calculate the number of elements in the given range.

Worst Case Complexity: If the BST is a long one with only right or left children and all the BST elements are in the range, we get the worst-case complexity which is O(n). Since the worst case is just a linked list, we can traverse it in O(n), so maybe we can make some adjustments and decrease the worst-case time complexity to O(n).

Mirror

To implement the mirror function, in the mirrorHelper function, I called the mirrorHelper function for both right and left subtrees while the root is not null. After both subtrees are mirrored, I swapped both trees by changing their pointers.

Worst Case Complexity: Since we mirror all node’s right and left children, we need to traverse all the nodes. Because of this worst-case complexity for the mirror function is O(n). Because the mirroring process requires us to visit every node, we cannot improve this function.