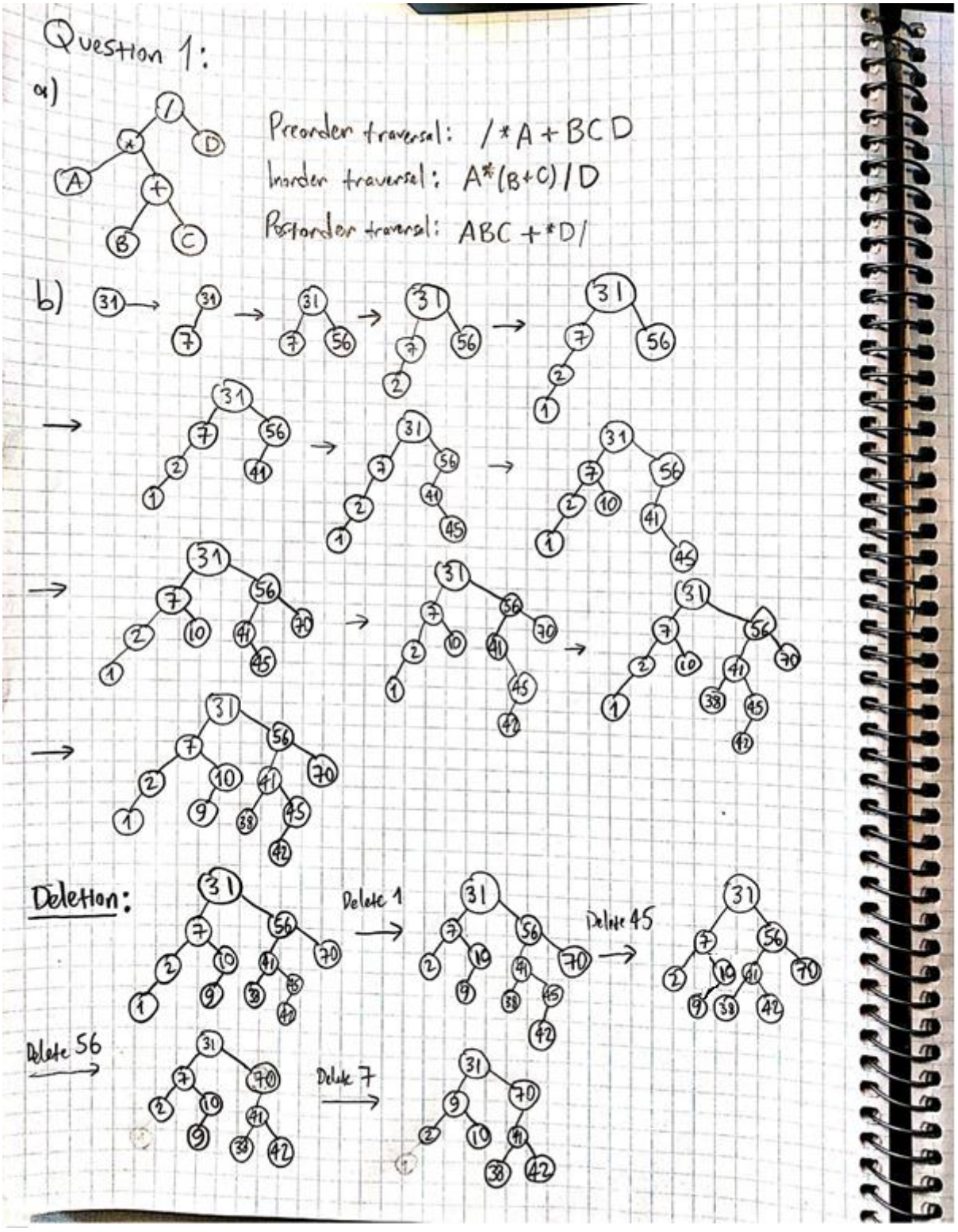
Efe Beydoğan

21901548

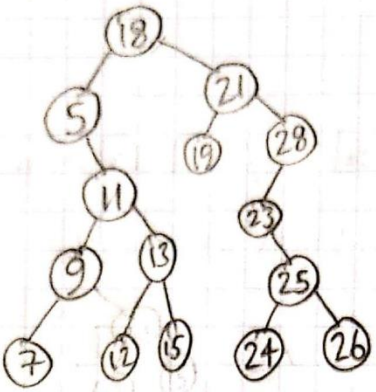
CS202 Section: 1

HW2 Report

Question 1:



c)



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

Question 3:

I used a Queue in the implementation of the levelorderTraverse function. In order to print the elements in the tree in a level order, first the root is enqueued, unless it is null. Then, within a while loop, initially the first element in the queue is dequeued and printed, then its right and left children are enqueued and the while loop continues until the queue is empty, meaning every element in the tree has been printed. The worst case time complexity of the levelorderTraverse function I wrote is O(n) since the function only traverses the given tree and goes through every element once, in level order. The function cannot be made asymptotically faster since the level order traverse is supposed to print every element once, thus it should visit every element at least once and hence the best complexity that can be achieved must be O(n).

The span function only increases the count if a node with value in the given interval is found. In order to avoid checking unnecessary elements in the tree, if a node with value less than the lower bound is encountered, the function returns and the right child pointer of the current node is passed to the function, since the elements in its left subtree will be smaller than the current node's item, meaning they will definitely be out of bounds. The same applies if a node with item bigger than the upper bound is encountered, in which case the function returns and passes the left child pointer of the current node to make sure it is not checking unnecessary elements in its right subtree. The worst case time complexity of the span function I wrote is O(n) since in the worst case, it has to check all the elements in the tree because they will all be in the given span. Since the function doesn't check unnecessary elements and only looks at elements that could be in the span, it cannot be made faster.

The mirror function I have written starts from the top and first interchanges the left and right child pointers of the root. Then, starting from the top of the root's left subtree, the function goes from the top to bottom, interchanging the left and right child pointers of every node. The same is done for the right subtree as well. The time complexity of the function is

O(n) since it traverses the whole tree and mirrors it. The mirror function is supposed to traverse the whole tree and thus cannot be made faster than O(n) since every node's children must be switched.