CS 202, Spring 2021

Homework 2 – Binary Search Trees

Arda Önal

21903350

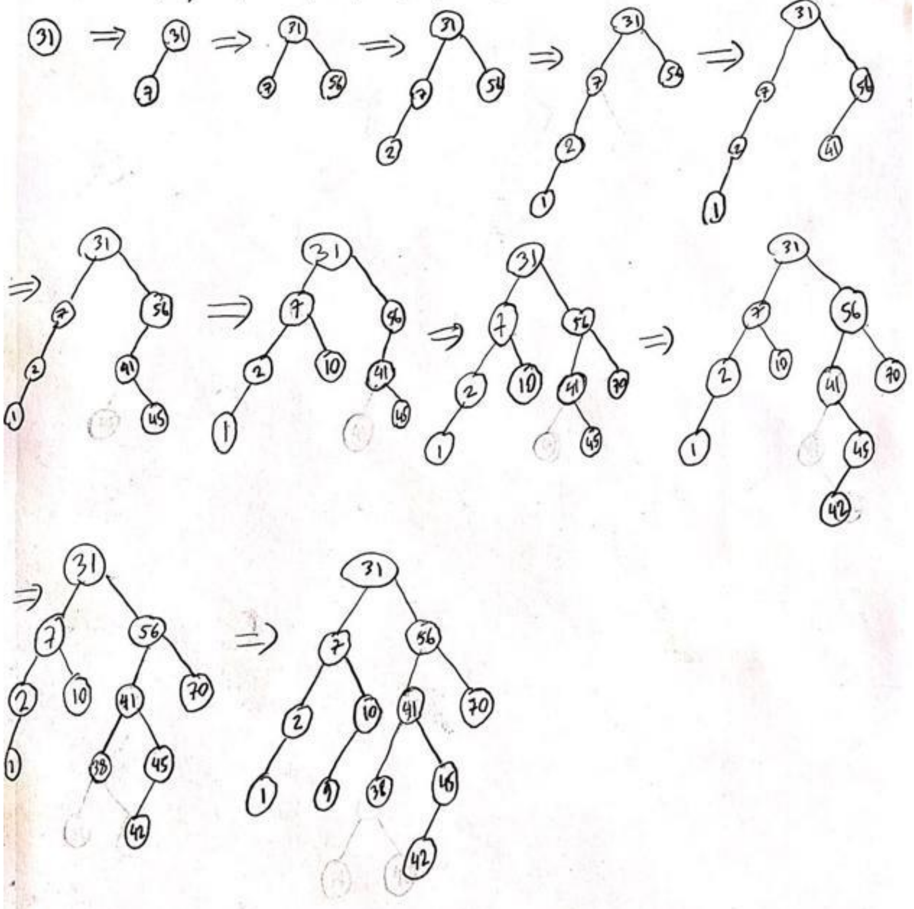
Section 3

Question 1:

Question 1:

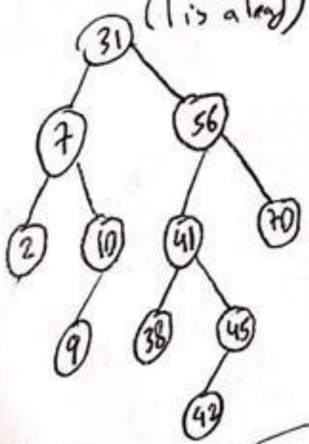
a) preorder: / * A + B C D   inorder: A * B + C / D   postorder: A B C + * D /

b) Insertion: 31, 7, 56, 2, 1, 41, 45, 10, 70, 42, 38, 9

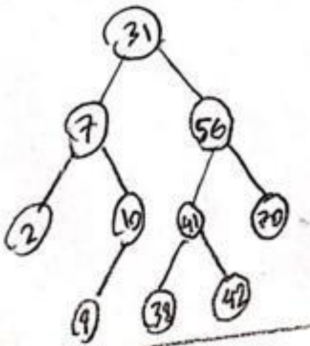


b) Deletion of 1, 45, 56, 7

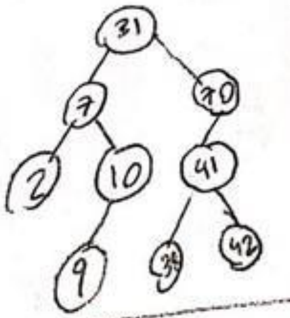
Tree after deleting 1; (1 is a leaf)



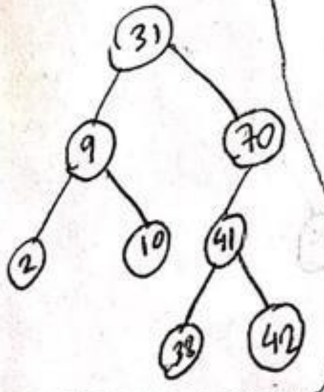
Tree after deleting 45! (45 has no right child)



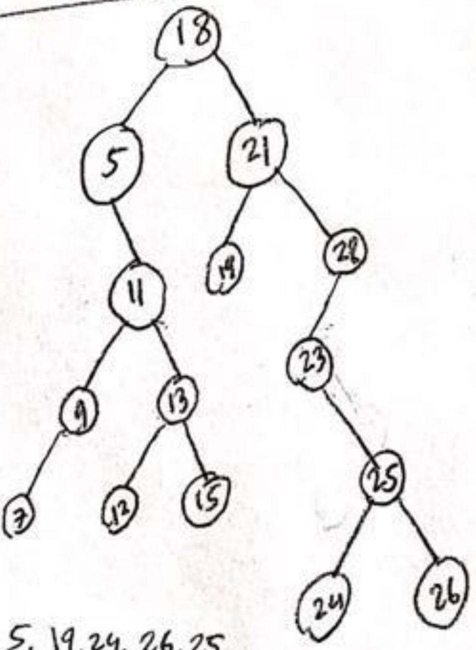
Tree after deleting 56! (56 has two children)



Delete 7;



c)



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

Question 3:

For the implementation of levelorderTraverse function, I have a function called "printLevel" and it is called number of levels times in the levelorderTraverse function. The printLevel function basically prints all nodes at a given level. It does this by going down from the root to the level and only printing when it gets to the level. In the function, it is written as "print if level equals one" but actually the level parameter is decremented each time go to a child so that the tree is traversed until we reach the level that we want to print. For the worst case, the printLevel has time complexity O(n) and levelorderTraverse function calls printLevel n times in the worst case therefore, the time complexity of my implementation is O(n^2). The level order traverse could have been implemented more efficiently with faster time complexity by the use of some data structures other than trees but the implementation would be much harder. For example if we use a queue, we could put the elements in a queue, and then dequeuer by printing. It is done by queueing the right child then the left child of the dequeued element. Hence it prints by going from left to right and root to leaf in the tree. It has a time complexity of O(n) which is much faster than my O(n^2) implementation.

For the implementation of span function, I used a count parameter and it basically is incremented when the value of the node is in between a and b. The rest of it regular tree traversal however, in order to avoid visiting unnecessary nodes, if the item is less than a, we go right and if the item is greater than b, we go left so that it skips nodes that already is known that they are not in the range [a,b]. This implementation has worst case time complexity O(n) because in the worst case, the "skipping" part is executed and every node is visited one by one. I believe this function cannot be implemented faster than worst case time complexity O(n) because the necessity of visiting every node is unavoidable.

For the implementation of mirror function, it basically swaps the left and right child of a node and tree is traversed recursively. Therefore, it swaps then, it goes to the left of the already swapped node and swaps the right and left child of the swapped node and the process is done for the right subtree so that the tree is fully mirrored. It has a worst case time complexity O(n) because when mirroring a tree we have to visit every node and visiting every node has O(n) time complexity. I believe this function cannot be implemented faster than worst case time complexity O(n) because the necessity of visiting every node is unavoidable.