CS 202

Spring 2019

Section: 1

Name: Hassan Raza

ID No: 21701811

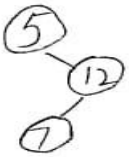
Homework #2 – Binary Search Trees

Q1(a) Insert 5, 12, 7, 1, 6, 3, 13, 2, 10, 11 in BST.

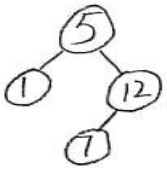
step 1:

step 2:

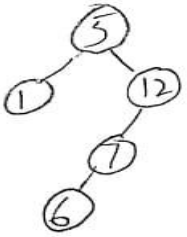
step 3:



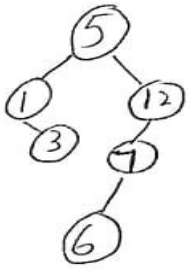
step 4:



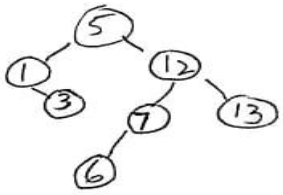
step 5:



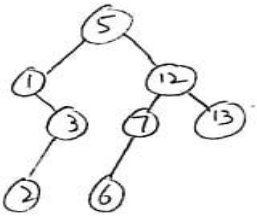
step6:



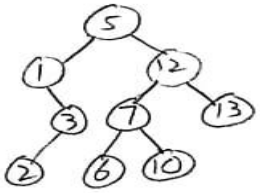
step 7:



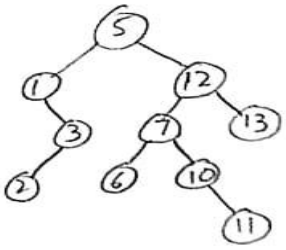
step 8:



step 9:



step 10:



Question 1(b): Preoreder, Inorder, Postorder traversals of BST:

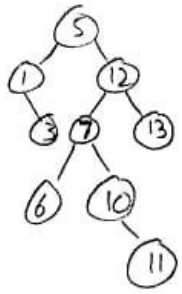
Preoreder: 5, 1, 3, 2, 12, 7, 6, 10, 11, 13

Inorder: 1, 2, 3, 5, 6, 7, 10, 11, 12, 13

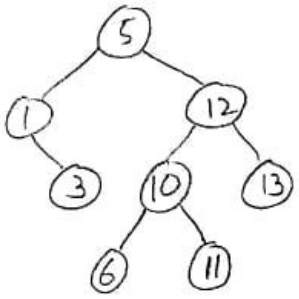
Postorder: 2, 3, 1, 6, 11, 10, 7, 13, 12, 5

Q 1 (c) Delete 2, 7, 5, 6, 11 from BST

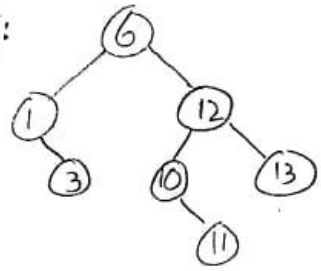
Delete 2:



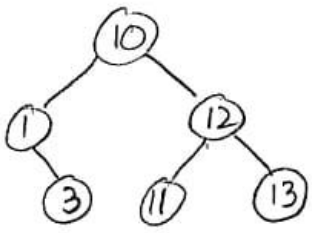
Delete 7:



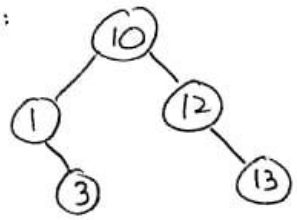
Delete 5:



Delete 6:



Delete 11:



Question 3:

insertItem:

In this method we go to place we need to insert item by going either left or right of node by comparing node with Item.

Worst Case Time Complexity: O(n), because we have to visit all nodes.

Average Case Time Complexity: O(log(n)), because on average at each step we will rule out half of elements as we go on other side from node.

Worst Case Space Complexity: O(1), we use constant space.

Average Case Space Complexity: O(1), we use constant space.

deleteItem:

In this method we go to place we need to delete Item by going either left or right of every node determined by comparison, once found there are four cases depending on them we do different things.

Worst Case Time Complexity: O(n), because we have to visit all nodes.

Average Case Time Complexity: O(log(n)), because on average at each step we will rule out half of elements as we go on other side from node.

Worst Case Space Complexity: O(1), we use constant space.

Average Case Space Complexity: O(1), we use constant space.

reterieveItem:

In this method we go to place we need to retrieve item from by going either left or right of each node by comparing node with Item.

Worst Case Time Complexity: O(n), because we have to visit all nodes.

Average Case Time Complexity: O(log(n)), because on average at each step we will rule out half of elements as we go on other side from node.

Worst Case Space Complexity: O(1), we use constant space.

Average Case Space Complexity: O(1), we use constant space.

inorderTraversal

In this method we go to each node and store its value in array. We do inorder traversal giving us an ascending ordered array.

Worst Case Time Complexity: O(n), because we have to visit all nodes.

Average Case Time Complexity: O(n) because we have to visit every node in every case

Worst Case Space Complexity: O(n), we are creating an array of n elements.

Average Case Space Complexity: O(n), we have to create an array of n elements in every case.

containsSequence:

This method looks if the given sequence is contained in tree. We go to next node by comparing last element of array with left child and first element of array with right child of node. At each node it compares if one of element in array is same as of node, in that case t checks if it is ordered from there. Suppose k is number of elements in array

Worst Case Time Complexity: O(n+k) = O(n) or O(k), depending on value of k. We have to visit all nodes to find the element contained in array after that just k visits for comparison.

Average Case Time Complexity: O(log(n) + k) = O(k) or O(log(n)),depending on value of k. We visit either left or right of node making complexity O(log(n)), and after finding a node we need to go over k nodes for comparison.

Worst Case Space Complexity: O(1), we use constant space.

Average Case Space Complexity: O(1) we use constant space.

countNodesDeeperThan:

In this method we count number of nodes deeper than given index. We traverse through all nodes keeping track of current level and if current is greater than index adding it to deeper nodes.

Worst Case Time Complexity: O(n), because we have to visit all nodes.

Average Case Time Complexity: O(n) because we have to visit all nodes in every case.

Worst Case Space Complexity: O(1), we use constant space.

Average Case Space Complexity: O(1), we use constant space.

maxBalancedHeight:

In this method do a post order traversal. We find the height of left and right subtrees at each level. If height difference is more than one we decrement height of that subtree by number so that difference becomes one. Height we get at the end is the height of balanced tree

Worst Case Time Complexity: O(n), because we have to visit all nodes checking there left and right subtree's height.

Average Case Time Complexity: O(n) because in every case we have to visit all nodes for comparison.

Worst Case Space Complexity: O(1), we use constant space.

Average Case Space Complexity: O(1), we use constant space.