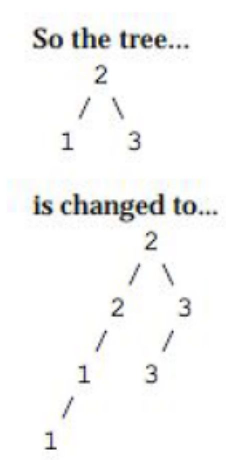
1. Implement following function in the Binary Search Tree class
    a. Constructor, copy constructor, destructor, assignment operator.
    b. BinarySearchTree(const vector<T> & sorted) – given a sorted array construct the balanced binary search tree. [Constructor overloading]
    c. void insertElement(const T &el)
    d. void removeElement(const T & el)
    e. const Node<T> * findElement(const T &el) const;
    f. const Node<T> * findMinimum() const;
    g. const Node<T> * findMaximum() const;
    h. const Node<T> * findLowestCommonAncestor(const T &el1, const T & el2) const – you should be able to do it better time than Binary Tree
    i. int size() const; - Returns number of elements in the bst.
    j. bool isEmpty() const; - If tree is empty or not
    k. void clear(); - Empty the tree
    l. const Node<T> * findInorderSuccessor(const T & el) const – Find inorder success of the given element.
    m. const BSTNode<T> * findInorderPredeccessor(const T & el) const – Find inorder predecessor of the given tree.
    n. void insertDuplicate() – For each node in the BST, create a new duplicate node and insert the duplicate as the left child of the original node.



    o. void updateNodesWithSumOfAllGreaterNodes() – Replace data of the each node with the sum of all greater nodes in a given BST.
    p. void printInRange(const T & K1, const T & K2) const – Print all nodes between range K1 and K2

   q. void createTreeFromPostOrder(const vector<T> & pre) – create tree from post order traversal.
   r. pair<Node<T> *, Node<T> *> findPairWithSum(const T & sum) – Find pair of node which sum to s.
      i. Find a solution with time complexity O(N)
      ii. Find a solution which uses maximum O(logN) extra space.

2. Suppose you are building an N node binary search tree with the values 1..N. How many structurally different binary search trees are there that store those values? For example, countTrees(4) should return 14, since there are 14 structurally unique binary search trees that store 1, 2, 3, and 4. Your code should not construct any actual trees; it's just a counting problem.
3. Given a BinaryTree check if it's a BST or not.
4. Convert a BinaryTree into a double linked list such that linear traversal of the linked list is same as in-order traversal of the tree.
5. Given a binary tree and a node of tree, print all nodes which are at distance k from the given node.
6. Given a binary tree, find its minimum depth. The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.