1. Implement following function in the GenericTree class. Always remember for recursive approach you would need two functions – a non-static public function which drives a static private function.
    a. copy constructor, destructor, assignment operator.
    b. void iterativeCreateTree()
    c. Void printIteratative() const {} – Use queue to print the tree
    d. Void printRecursive() const - Print Recursively
    e. int getHeight() const {} – recursive compute height of the tree.
    f. const Node<T> * findNodeWithLargestData() const {} – return data with largest data.
    g. void printAllElementsAtDepthK(int K) const {} – Print all nodes which are at depth K.
    h. int countNodesWithDataGreaterThanRoot() const {} – Return count of all nodes which have data greater than root.
    i. int countLeafNodes() const {} – Return count of all leave nodes in the generic tree.
    j. const Node<T> * nodeWithSecondLargestData() const – return address of the node which has the second largest data
    k. bool operator==(const GenericTree & T) const {} – Return true if the two trees are structurally identical i.e. they are made up of nodes with same value arranged in the same way.
    l. const Node<T> * findSomeSpecialNode() const {} – Return address of a node for which sum of data of all its children and the node itself is maximum in the generic tree.
    m. const Node<T> * findJustLargerThanK(const T & K)const; – Return address of the node just higher to K.
    n. void iterativePostOrderTraversal() const;
    o. void printPostOrderRecursive() const;
    p. void printLevelOrder() const; - Print Each level in a new line
    q. void printZigZagLevelOrder() const;- Print the zig zag order i.e print level 1 from left to right, level 2 from right to left and so on. This means odd levels should get printed from left to right and even levels should be printed from right to left. Each level should be printed at a new line.
    r. bool isThereARootToLeafPathWithSumK(const T & K)const; – return true if there is a root to leaf path with total sum K
    s. const Node<T> * findLowestCommonAncestor(const T & el1, const T

& el2) const;– Return address of LCA of el1 & el2. If they don't exist return NULL