**Range Sum of BST:**

Return the sum of values of all nodes with value between L and R

Touch every node adding node vals if within a range

public int rangeSumBST(TreeNode root, int L, int R) {

if (root == null) return 0;

int val = 0;

int left = (root.val > L) ? rangeSumBST(root.left, L, R):0;

int right =(root.val < R) ? rangeSumBST(root.right, L, R) : 0;

if(root.val >=L && root.val<=R)

val+= root.val;

return left+right+val;

}

Iterative is a stack<TreeNode>, checking value before adding nodes to stack

**Merge Two Binary Trees**

Add values of 2nd node to first

Touch every node in 2 trees adding t2 node val to t1 node val

public TreeNode mergeTrees(TreeNode t1, TreeNode t2) {

if (t1 == null) return t2;

if (t2 == null) return t1;

t1.val += t2.val;

t1.left = mergeTrees(t1.left, t2.left);

t1.right = mergeTrees(t1.right, t2.right);

return t1;

}

Iterative is a stack<TreeNode[]>, checking nullality ,value before adding nodes to stack

**Search in a Binary Search Tree**

Return the subtree of node that equals val

Touch log N nodes to find a node

public TreeNode searchBST(TreeNode root, int val) {

if (root==null) return null;

if (root.val==val)return root;

if(root.val > val)

return searchBST(root.left,val);

else

return searchBST(root.right,val);

}

Iterative is while loop (while(root != null && root.val != val))…

**Increasing Order Search Tree**

Rearrange tree inorder so nodes all have right children only

Touch every node adding to list using inorder and creating new tree from list

**Sum of Root To Leaf Binary Numbers**

Add values(0 or 1) in each root-to-leaf path by binary manipulation

Touch every node via dfs summing binary values from root to each leaf

void sumRootToLeafWorker(Pair<TreeNode, Integer> item) {

TreeNode node = item.getKey();

if(node.left==null && node.right==null) sumrtl+=item.getValue() ;

if(node.left!=null) sumRootToLeafWorker(new Pair(node.left,item.getValue() << 1 | node.left.val));

if(node.right!=null) sumRootToLeafWorker(new Pair(node.right,item.getValue()<<1 | node.right.val));

}

Iterative is stack<TreeNode>, …

**Univalued Binary Tree**

Has every node same value?

Touch every node using dfs to see if all same value

boolean isUnivalTreeWorker(TreeNode root, int value) {

if (root == null) return true;

return (root.val == value && isUnivalTreeWorker(root.left, value) && isUnivalTreeWorker(root.right, value));

Iterative is stack<TreeNode>, …

}

**Maximum Depth of Binary Tree**

Return longest root-to-leaf path

Touch every node using DFS to calc depth of root to each leaf and store max of them

public int maxDepth(TreeNode root) {

if (root == null) return 0;

var left = maxDepth(root.left);

var right = maxDepth(root.right);

return Math.max(left, right) + 1;

}

Iterative is a queue<TreeNode>, using bfs to add nodes by level and summing up a level count

**Invert Binary Tree** ie 3,2,1 = 3,1,2…

Switch left and right nodes

Touch every node using dfs swapping each nodes left and right children

public TreeNode invertTree(TreeNode root) {

if (root == null) return null;

var right = invertTree(root.right);

var left = invertTree(root.left);

root.left = right;

root.right = left;

return root;

}

Iterative is a stack<TreeNode>, swapping value before adding nodes to stack

**Leaf-Similar Trees**

Hve 2 nodes same order of leaf values?

Touch every node of 2 trees using dfs comparing if their leafs equal value and order

while (!left.isEmpty() && !right.isEmpty())

if (getNodeValue(left) != getNodeValue(right)) return false;

return (left.isEmpty()&& right.isEmpty());

}

int getNodeValue(Stack<TreeNode> stack) {

while (true) {

var current = stack.pop();

if (current.left == null && current.right == null) return current.val;

if (current.right != null) stack.push(current.right);

if (current.left != null)stack.push(current.left);

}

}

Iterative is a 2 stacks, popping a node from each and checking both leafs values are same. \*\*\*

while (!left.isEmpty() && !right.isEmpty()) {

if (getNodeValue(left) != getNodeValue(right)) //has while true loop that

return false; // returns if leaf, otherwise adds node to stack

}

**Average of Levels in Binary Tree**

Return average of each level as list

Touch every node using BFS to calc average of each level storing in a list

Iterative is a queue<TreeNode>, using bfs to add vals average by level into list

**Trim Binary Search Tree**

Remove nodes within certain range from tree

Touch every node using dfs, removing nodes that fall out of range

public int trim Bst(TreeNode root, int low, int high) {

if (root == null) return root;

if (root.val > high) return trimBST(root.left, low, high);

if (root.val < low) return trimBST(root.right, low, high);

root.left = trimBST(root.left, low, high);

root.right = trimBST(root.right, low, high);

return root;

}

Iterative uses 3 loops, 1 find root, 2 remove invalid nodes from left, 3 remove from right..\*\*\*

**Convert Sorted Array to Binary Search Tree**

BST tree depths musn’t differ by >1

Get middle of list and repeat using left and right of list adding to new node

… return sortedArrayToBSTWorker(nums, 0, nums.length-1);

}

TreeNode sortedArrayToBSTWorker(int[] nums, int low, int high) {

if (low <= high) {

var mid = low + (high-low)/2;

TreeNode node = new TreeNode(nums[mid]);

node.left = sortedArrayToBSTWorker(nums, low, mid - 1);

node.right = sortedArrayToBSTWorker(nums, mid + 1, high);

return node;

}

return null;

}

Iterative is a stack<TreeNode, left, right> that adds kids to nodes left and right along with adding the new nodes to stack..

**Two Sum IV - Input is a BST**

Do 2 nodes in tree sum to certain value?

Touch every node using b/dfs, adding nodes val to set and checking if node – k exists in set

…return find(root, k, set);

}

public boolean find(TreeNode root, int k, Set < Integer > set) {

if (root == null) return false;

if (set.contains(k - root.val)) return true;

set.add(root.val);

return find(root.left, k, set) || find(root.right, k, set);

}

Iterative is a a set and a q. Add all nodes to q and set seeing if (target - node.val) exists in set. Actually use 2 queues and 0 set (2 O(log n) is < than 1 O(n) \*\*\* (see top voted) ^^^^

**Construct String from Binary Tree**

Return string of tree with ( ) surrounding child nodes.

Touch every node using b/dfs, adding nodes val to set and checking if node – k exists in set

String tree2str(TreeNode t) {

**if** (t == **null**) **return** "";

**if** (t.left == **null** && t.right == **null**) **return** t.val + "";

**if** (t.right == **null**) **return** "" + t.val + "(" + *tree2str*(t.left) + ")";

**return** "" + t.val + "(" + *tree2str*(t.left) + ")(" + *tree2str*(t.right) + ")";

}

Iterative uses a set, stack and stringbuilder. Uses peek() and adds to set if new or else pops off stack. So hits each node twice \*\*\*

**Binary Tree Level Order Traversal II**

Return levels of tree in reverse

Touch every node using bfs, adding array of level nodes val to stack, before popping each stack item off into a list

…void levelMaker(List<List<Integer>> list, TreeNode root, int level) {

if(root == null) return;

if(level >= list.size()) list.add(0, new LinkedList<Integer>());

levelMaker(list, root.left, level+1);

levelMaker(list, root.right, level+1);

list.get(list.size()-level-1).add(root.val);

}

Iterative uses BFS and q and adds nodes on each level to stack<int[]>. Then remove from stack to list.

**Minimum Absolute Difference in BST**

Return min between 2 nodes

Touch log N nodes using dfs, compring delta between adjacent node values

int min = Integer.MAX\_VALUE;

Integer prev = null;

public int getMinimumDifference(TreeNode root) {

if (root == null) return min;

getMinimumDifference(root.left);

if (prev != null) min = Math.min(min, root.val - prev);

prev = root.val;

getMinimumDifference(root.right);

return min;

}

Iterative is Inorder with !stack.isEmpty() or current!=null.

**Same Tree**

Are 2 trees exact same

Touch every node in both trees using dfs, returning false if nodes aren’t same value

public boolean isSameTree(TreeNode p, TreeNode q) {

if (p == null && q == null)return true;

if (p == null || q == null)return false;

return p.val == q.val && isSameTree(p.left, q.left) && isSameTree(p.right, q.right);

}

Iterative is just 2 stacks comparing both nodes. If you have 2 stacks you can always use 1 Queue instead.. ^^^^

**Binary Tree Paths**

Return all root to leaf paths

Touch every node using dfs, using a string builder and add paths to list

binaryTreePathsWorker(TreeNode root, List<String> result, StringBuilder sb) {

**if** (root == **null**)**return**;

**var** len = sb.length();

sb.append(root.val); // s = s + root.val + "->";

**if** (root.left == **null** && root.right == **null**)

result.add(sb.toString());// s.substring(0, s.length() - 2));

**else** {

sb.append("->");

*binaryTreePathsWorker*(root.left, result, sb);

*binaryTreePathsWorker*(root.right, result, sb);

}

sb.setLength(len);

}

**Iterative**

**Cousins in Binary Tree**

Are 2 values on same level but have different parent?

Touch every node in both trees using dfs, …

public boolean isSameTree(TreeNode p, TreeNode q) {

}

**Sum of Left Leaves**

Return sum of left leafs

Touch every node in both trees using dfs summing left leaf

public int sumOfLeftLeaves(TreeNode root) {

if (root == null) return 0;

int l = sumOfLeftLeaves(root.left);

int r = sumOfLeftLeaves(root.right);

return l + r + (root.left != null && root.left.left == null && root.left.right == null? root.left.val : 0);

}

**LCA of BST**

Return lca of 2 passed in values

Touch every node (worst), log N nodes otherwise, returning parent if 1 value is > and other is < than root

public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

if (root == null) return null;

if ((p.val <= root.val && q.val >= root.val) || (p.val >= root.val && q.val <= root.val))

return root;

if (p.val <= root.val)

return lowestCommonAncestor(root.left, p, q);

else

return lowestCommonAncestor(root.right, p, q);

}

**Closest Binary Search Tree Value**

Return node value closest to target(a double)

Touch every node calculating closest val using Min

public int closestValue(TreeNode root, double target) {

List<Integer> nums = new ArrayList();

inorder(root, nums); … the usual inorder method adding values to above list

return Collections.min(nums, new Comparator<Integer>() {

@Override

public int compare(Integer o1, Integer o2) {

return Math.abs(o1 - target) < Math.abs(o2 - target) ? -1 : 1;

}

});

}

**Diameter of Binary Tree**

Return longest path of 2 nodes

Touch every node saving max of sum of length of a nodes left and right paths

int max\_dobt;

int diameterOfBinaryTreeWorker(TreeNode root) {

if (root == null)return 0;

var l = diameterOfBinaryTreeWorker(root.left);

var r = diameterOfBinaryTreeWorker(root.right);

max\_dobt = Math.max(max\_dobt, l + r);

return Math.max(l, r) + 1;

}

**Symmetric Tree**

Is 2nd tree within 1st tree?

Touch every node comparing left and right vals of 2 nodes to see if they equal

….

private boolean isSymmetricWorker(TreeNode left, TreeNode right) {

if (left == null && right == null) return true;

if (left == null || right == null) return false;

var l = isSymmetricWorker(left.left, right.right);

var r = isSymmetricWorker(left.right, right.left);

return l && r && left.val == right.val;

}

**Subtree of Another Tree**

Return sum of left leafs

Touch every node in both trees using dfs to find if 2nd tree == part of 1st tree

public boolean isSubtree(TreeNode s, TreeNode t) {

if (s == null && t == null) return true;

if (s == null || t == null) return false;

if (isSubtreeWorker(s, t))

return true;

else

return isSubtree(s.left, t.left) || isSubtree(s.right, t.right);

}

boolean isSubtreeWorker(TreeNode t1, TreeNode t2) {

if (t1 == null && t2 == null) return true;

if (t1 == null || t2 == null) return false;

return t1.val == t2.val && isSubtree(t1.left, t2.left) && isSubtree(t1.right, t2.right);

}

**Balanced Binary Tree**

Does height differ by max 1 level?

Touch every node using dfs calculating height of each r2l path

public bool isBalanced(TreeNode \*root) {

return dfsHeight (root) != -1;

}

int dfsHeight (TreeNode root) {

if (root == NULL) return 0;

int leftHeight = dfsHeight (root.left);

if (leftHeight == -1) return -1;

int rightHeight = dfsHeight (root.right);

if (rightHeight == -1) return -1;

if (abs(leftHeight - rightHeight) > 1) return -1;

return max (leftHeight, rightHeight) + 1;

}

**Path Sum**

Does r2l path exist whose values equal target?

Touch every node using dfs adding values from root to leaf

public boolean hasPathSum(TreeNode root, int sum) {

if (root == null) return false;

sum -= root.val;

if (root.left == null && root.right == null && sum == 0) return true;

return hasPathSum(root.left, sum) || hasPathSum(root.right, sum);

}