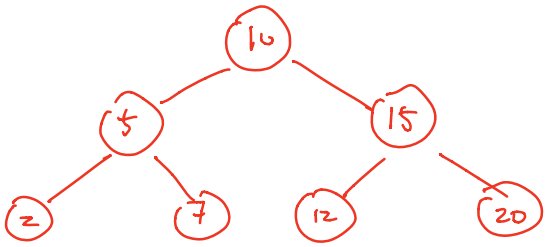
Binary Tree: at most 2 children node. (left & right)



Tree Traversal:

pre-order: self → left → right.

10, 5, 2, 7, 15, 12, 20

in-order: left → self → right.

2, 5, 7, 10, 12, 15, 20

post-order: left – right – self

2, 7, 5, 12, 20, 15, 10.

Balanced: height of the left & right subtrees of every node differ by ≤ 1.

if a tree has n nodes, & is balanced.

then its height = $O(\log_2 n)$.

if not balanced. height = $O(n)$.

Completed: every level (except the last) is completely filled.

as far as left possible.

if complete, must be balanced.

Intuition: want to get value for a problem w/ size n.

recursion: solve problem of size n-1.

example: getHeight().

Q1: Is a tree balanced? subtrees height differ by 1 or less.

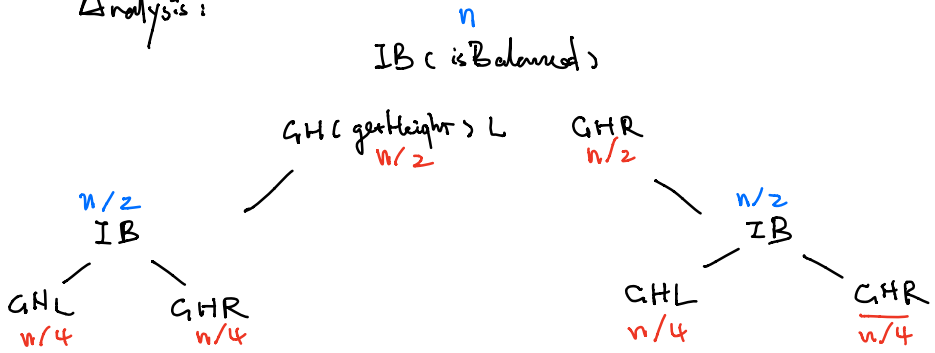
Soln:

Base: null — true.

recursion: if diff > 1. false.

return isBalanced(left) && isBalanced(right)

Analysis:



each level: O(n)

level: O(logn).

Total: O(nlogn).

Q2. Judge if a BST is symmetric.

check both left & right. isSymmetric(L, R).

Base: #1. L = null && R = null → true

#2. L = null || R = null → false.

#3. L.val != R.val → false.

recursion: return isSymmetr(L.L, R.R).

&& isC(L.R, R.L).

analysis : Time: O(n/2) ~ O(n).

Space: O(height)

Iterative way: Queue. ☆

add 2 roots. & while(!Q.isEmpty()).

poll 2 nodes.

check base cases.

add n1.left & n2.right.

add n1.right & n2.left.

Q3. if same Structure (omitted). Todo.

Q4. Determine if a BST is valid.

valid: left only contains nodes w/ keys less than itself.

right ... greater than

both left & right subtrees also BST

Soln 0: #1 inorder traversal and store

#2 iterate & check Arr[i] ≤ Arr[i+1].

Primitive but bad in space consumption.

Soln 0 better: inorder traversal. when print new.

compare w/ previous & check.

Soln 1: Set min & max boundary for each layer (level)

whenever

Assumption: NO Duplicates

Go down to the Left: update MAX;

Go down to the right: update MIN

Base: root == null → true.

Recursion: if root.val ∉ (min, max) → false.

return recursion(root.left, min, root.val)

&& recursion(root.right, root.val, max)

Note: function signature (int min, int max).

assume Integer.MIN or MAX_VALUE is not key.

IF NO SUCH ASSUMPTION !.

use (Integer min, Integer max)

and pass (null, null) to handle edge cases.

Note: Recursion in Tree.

1. pass value from top to bottom

and then pass back from bottom to top

eg: validate BST.

2. Only pass values from bottom to top.

eg: getHeight()

isBalanced()

isSymmetric()

Assign the value of each node to be the total # of nodes that belong to its left subtree.

Q5. Print BST keys in the given range (Unique keys).

AKA. Trim a binary Tree. Given low & high.

Keep the relative order of the subtree.

Notes: instead of returning boolean. need to cut off nodes that are not [Low, high].

BST might NOT be valid. Strictly.

Determine which direction to go.

if root > lower: go down left.

within range. print

if root < upper: go down right

If we want to Trim the tree.

First, need to locate a node that's within the range.

if root > high : root → root.left

if root < low : root → root.right

return root (within range)

Cut off Situation:

if root == null. return null

if root < low. return recursion (root.right)

if root > high. return recursion (root.left).

Divide into small problems: (once we found a node in range)

then determine directions to go.

~~Go left: if root > low~~

root.left = recursion(root.left, low, high)

~~Go right: if root < high.~~

root.right = recursion(root.right, low, high)

At last return root (restructured)