Tree

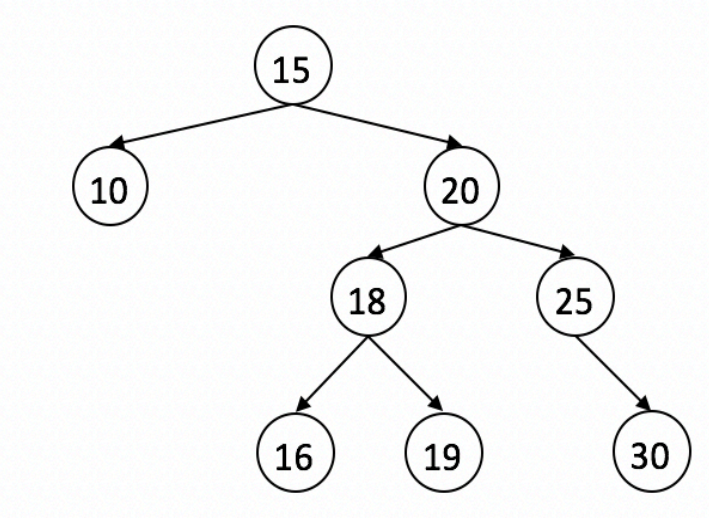
Height

Height of node

The height of a node is the number of edges on the longest path between that node and a leaf.

Height of tree

The height of a tree is the height of its root node.



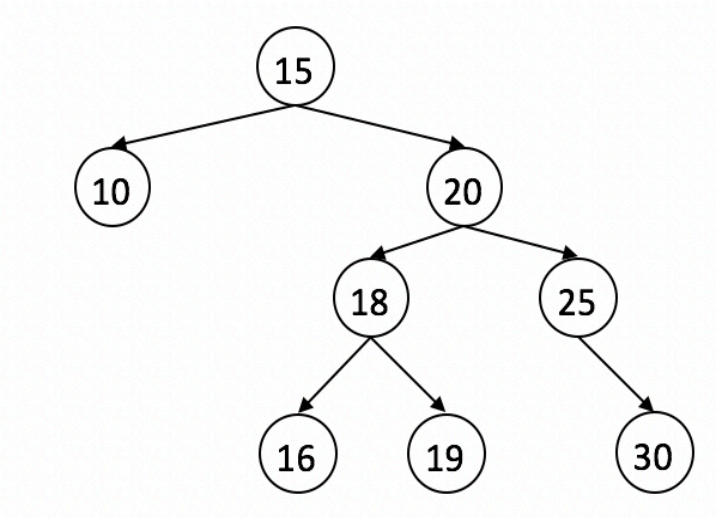
Height of node “20” is 2

Height of tree is 3

Depth

Depth

The depth of a node is the number of edges from the tree's root node to the node.

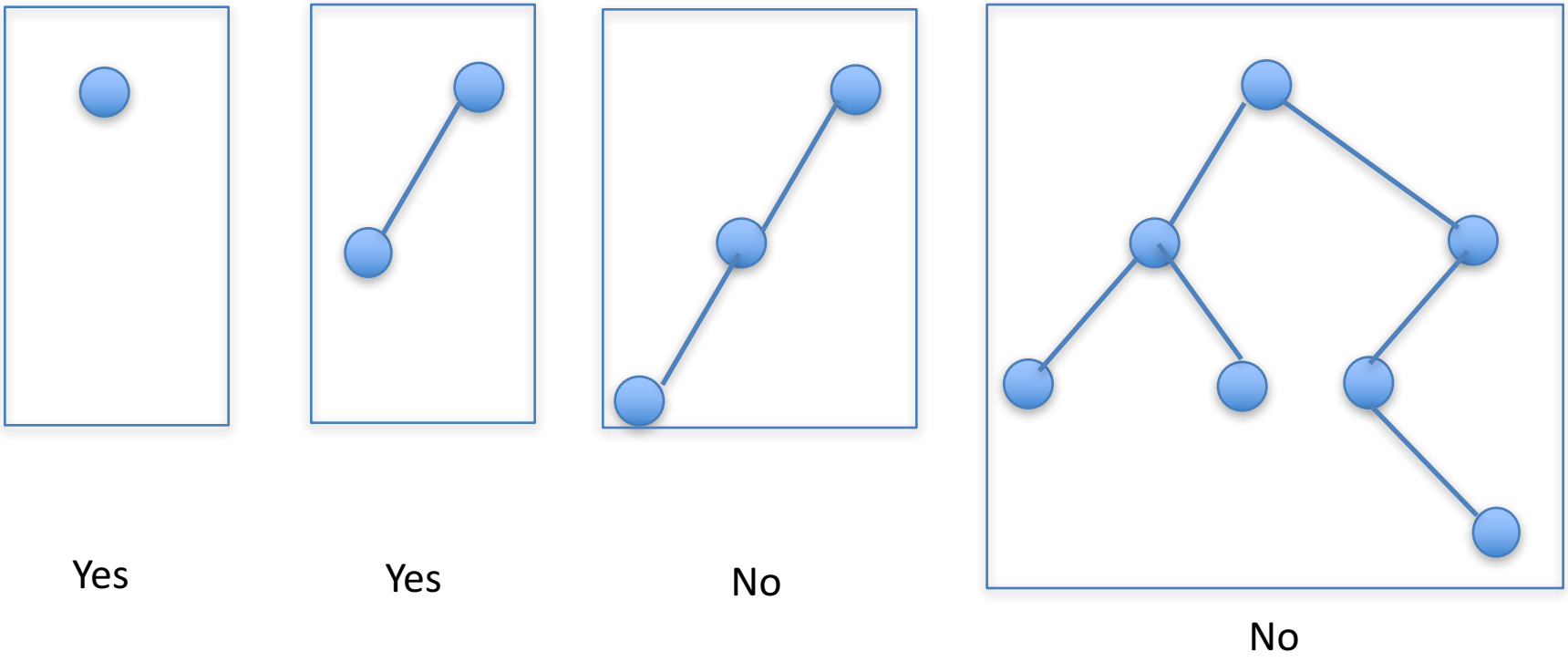


Depth of node “20” is 1

Balanced Trees

A binary tree is called a height balanced binary if it satisfies the following condition:

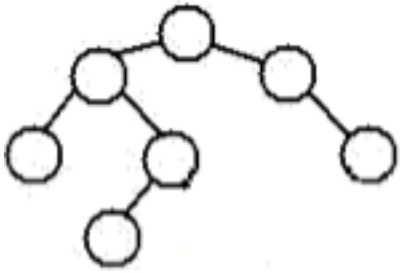
For all nodes of the tree, absolute difference between heights of left sub-tree and right sub-tree Is not greater than 1.



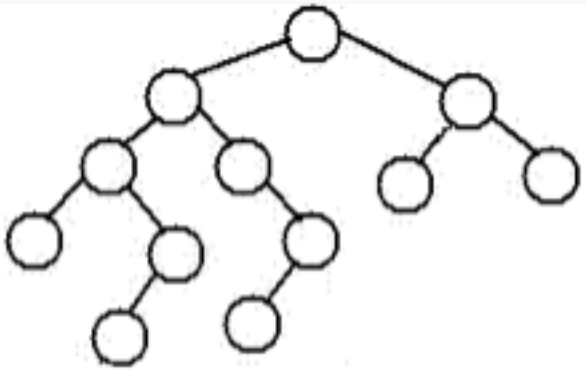
Yes Yes No No

Check if a tree is Balanced

Consider a height-balancing scheme where following conditions should be checked to determine if a binary tree is balanced.
An empty tree is height-balanced. A non-empty binary tree T is balanced if:
1) Left subtree of T is balanced
2) Right subtree of T is balanced
3) The difference between heights of left subtree and right subtree is not more than 1.



A height-balanced Tree



Not a height-balanced tree

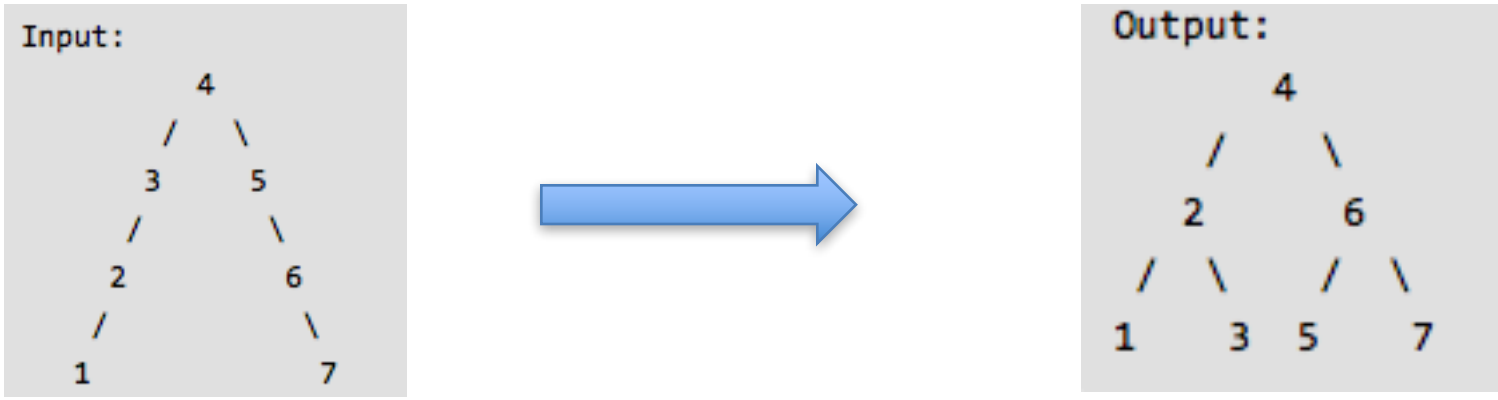
Cite: http://www.geeksforgeeks.org/how-to-determine-if-a-binary-tree-is-balanced/

Implementation

```
/*  The function Compute the "height" of a tree. Height is the
    number of nodes along the longest path from the root node
    down to the farthest leaf node.*/
int height(struct node* node)
{
   /* base case tree is empty */
   if(node == NULL)
       return 0;

   /* If tree is not empty then height = 1 + max of left
      height and right heights */
   return 1 + max(height(node->left), height(node->right));
}
```

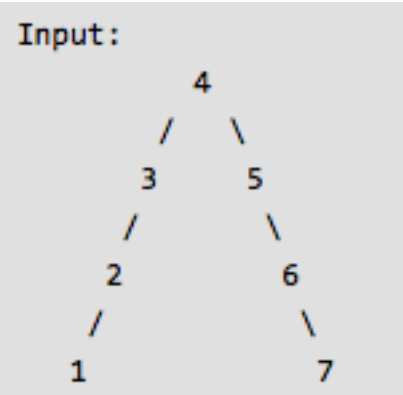
```
/* The function returns true if root is balanced else false
   The second parameter is to store the height of tree.
   Initially, we need to pass a pointer to a location with value
   as 0. We can also write a wrapper over this function */
bool isBalanced(struct node *root, int* height)
{
  /* lh --> Height of left subtree
     rh --> Height of right subtree */
  int lh = 0, rh = 0;

  /* l will be true if left subtree is balanced
    and r will be true if right subtree is balanced */
  int l = 0, r = 0;

  if(root == NULL)
  {
    *height = 0;
     return 1;
  }

  /* Get the heights of left and right subtrees in lh and rh
    And store the returned values in l and r */
  l = isBalanced(root->left, &lh);
  r = isBalanced(root->right,&rh);

  /* Height of current node is max of heights of left and
     right subtrees plus 1*/
  *height = (lh > rh? lh: rh) + 1;

  /* If difference between heights of left and right
     subtrees is more than 2 then this node is not balanced
     so return 0 */
  if((lh - rh >= 2) || (rh - lh >= 2))
    return 0;

  /* If this node is balanced and left and right subtrees
    are balanced then return true */
  else return l&&r;
}
```

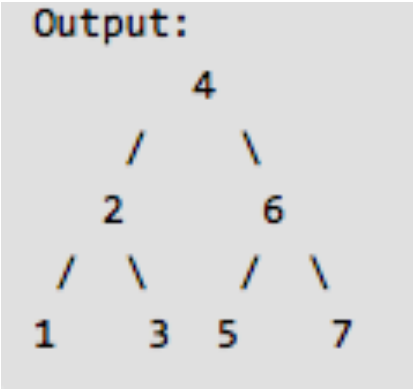
Convert a Imbalanced to Balanced



Implementation

1. Traverse given BST in inorder and store result in an array.
2. Get the Middle of the array and make it root. Recursively do same for left half and right half. a) Get the middle of left half and make it left child of the root created in step 1. b) Get the middle of right half and make it right child of the root created in step 1.





InOrder Traverse: 1-2-3-4-5-6-7