Agenda:
- Construct a Binary Tree, given its Preorder and Inorder traversals (discuss also for Postorder and Inorder).
- Iterative inorder traversal
- Introduction to Binary Search Trees
    - Discuss properties
    - Inorder traversal always sorted
- Searching a node in a BST
- Inserting a node in a BST
- Deleting a node from a BST
- How to check whether a given tree is a BST or not?
- Sorted array to BST
- Merge two BSTs (Just explain the approach using iterative Inorder)

Construct Tree from Inorder & Preorder

Given 2 Arrays of Inorder and preorder traversal. Construct a tree and print the Postorder traversal.
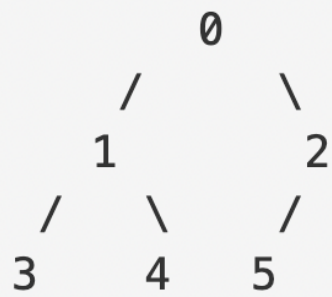
```
Input:
N = 6
inorder[] = {3 1 4 0 5 2}
preorder[] = {0 1 3 4 2 5}
Output: 3 4 1 5 2 0
Explanation: The tree will look like
        0
      /   \
     1     2
    / \   /
   3   4 5
```

Iterative inorder traversal

Given a binary tree, return the inorder traversal of its nodes values.

NOTE: Using recursion is not allowed.

```
      1
     / \
    6   2
       /
      3
```

```
[6, 1, 3, 2]
```

Introduction to Binary Search Trees

Binary Search Tree is a node-based binary tree data structure that has the following properties:
- The left subtree of a node contains only nodes with keys lesser than or equal to the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
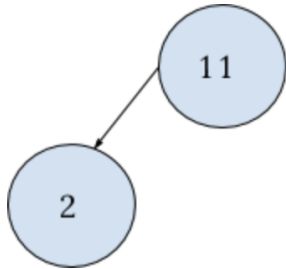- The left and right subtree each must also be a binary search tree.

There must be no duplicate nodes.

The basic purpose and idea behind such a data structure are to have a storing mechanism that provides a way for efficient sorting and searching operations.
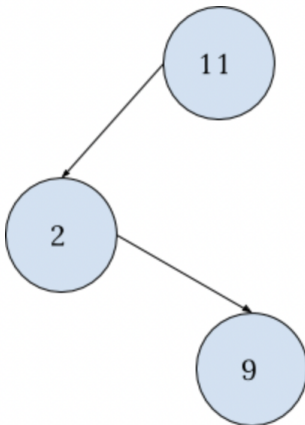
Inserting a Node into BST
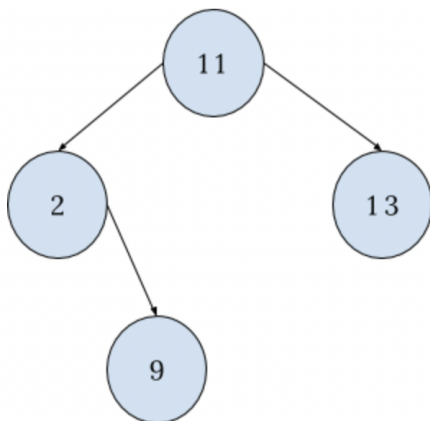
Any newly inserted node in a BST is always a root node.
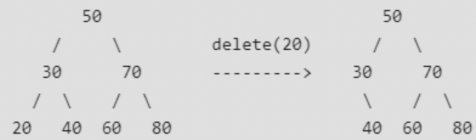
For example:

Insert 9:

Insert 13:

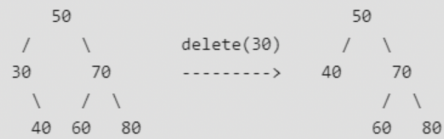# Deleting a Node from BST
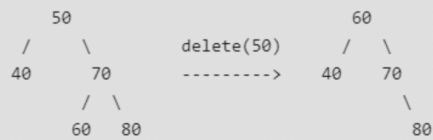
There are 3 possible cases:

1. *Node to be deleted is leaf:* Simply remove from the tree.

```
            50                              50
          /    \         delete(20)       /   \
        30      70      --------->       30     70
       /  \    /  \                        \   /  \
     20   40  60   80                      40 60   80
```

2. *Node to be deleted has only one child:* Copy the child to the node and delete the child.

```
            50                              50
          /    \         delete(30)       /   \
        30      70      --------->       40     70
          \    /  \                            /  \
          40  60   80                         60   80
```

3. *Node to be deleted has two children:* Find inorder successor of the node. Copy contents of the inorder successor to the node and delete the inorder successor. Note that inorder predecessor can also be used.

```
            50                              60
          /    \         delete(50)       /   \
        40      70      --------->       40     70
               /  \                               \
              60   80                              80
```
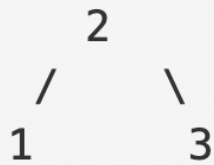
How to check whether a given tree is a BST or not?

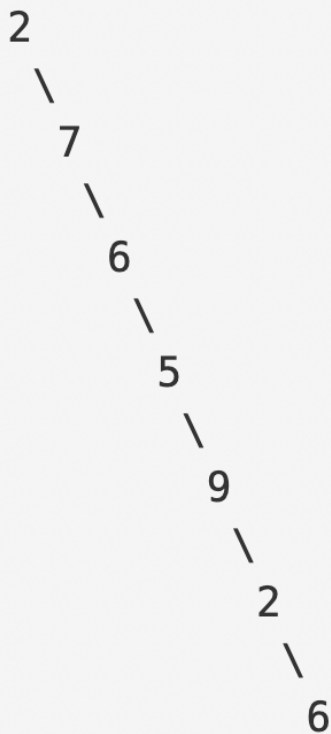Given the root of a binary tree. Check whether it is a BST or not.
Note: We are considering that BSTs can not contain duplicate Nodes.

```
Input:
    2
   /   \
  1     3
Output: 1
```

```
Input:
    2
     \
      7
       \
        6
         \
          5
           \
            9
             \
              2
               \
                6
Output: 0
```

Sorted array to Height Balanced BST

Given an integer array `nums` where the elements are sorted in **ascending order**, convert *it to a **height-balanced** binary search tree.*

A **height-balanced** binary tree is a binary tree in which the depth of the two subtrees of every node never differs by more than one.

Output: Preorder traversal of the returned BST

```
Input: nums = {1,2,3,4,5,6,7}
Ouput: {4,2,1,3,6,5,7}
Explanation:
The preorder traversal of the following
BST formed is {4,2,1,3,6,5,7} :
         4
        / \
       2   6
      / \ / \
     1  3 5  7
```