Table of Contents

Check if a binary tree is subtree of another binary tree | Set 2

Check if a Binary Tree (not BST) has duplicate values

Check if a Binary Tree contains duplicate subtrees of size 2 or more

Serialize and Deserialize a Binary Tree

Construct BST from given preorder traversal | Set 2

Construct BST from given preorder traversal | Set 1

A program to check if a binary tree is BST or not

Print the longest leaf to leaf path in a Binary tree

Print path from root to a given node in a binary tree

Print root to leaf paths without using recursion

Print the nodes at odd levels of a tree

Print all full nodes in a Binary Tree

Print nodes between two given level numbers of a binary tree

Print nodes at k distance from root

Print Ancestors of a given in Binary Tree

Check if a binary tree is subtree of another binary

A program to check if a binary tree is BST or not

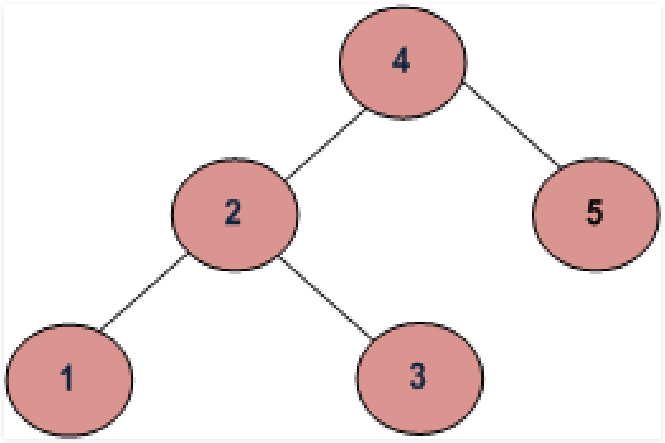
Difficulty Level : Medium • Last Updated : 18 Feb, 2022

A binary search tree (BST) is a node based binary tree data structure which has the following properties.

- The left subtree of a node contains only nodes with keys less than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- Both the left and right subtrees must also be binary search trees.

From the above properties it naturally follows that:

- Each node (item in the tree) has a distinct key.



Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

Skip to content

AVL Tree | Set 1 (Insertion)

Binary Search Tree | Set 2 (Delete)

C++

```
int isBST(struct node* node)
{
  if (node == NULL)
    return 1;

  /* false if left is > than node */
  if (node->left != NULL && node->left->data
    return 0;

  /* false if right is < than node */
  if (node->right != NULL && node->right->dat
    return 0;

  /* false if, recursively, the left or right
  if (!isBST(node->left) || !isBST(node->rig
    return 0;

  /* passing all that, it's a BST */
  return 1;
}


// This code is contributed by shubhamsingh1(
```

C

```
int isBST(struct node* node)
{
  if (node == NULL)
    return 1;

  /* false if left is > than node */
  if (node->left != NULL && node->left->data
    return 0;

  /* false if right is < than node */
  if (node->right != NULL && node->right->dat
    return 0;

  /* false if, recursively, the left or right
  if (!isBST(node->left) || !isBST(node->rig
    return 0;

  /* passing all that, it's a BST */
  return 1;
}
```

Java

```
boolean isBST(Node node)
{
    if (node == null)
        return true;

    /* False if left is > than node */
    if (node.le                 ode.left.data :
```

Skip to content

```
    /* False if, recursively, the left or rig
    if (!isBST(node.left) || !isBST(node.righ
        return false;

    /* Passing all that, it's a BST */
    return true;
}

// This code is contributed by shubhamsingh1(
```

Python3

```
def isBST(node):
    if (node == None):
        return 1

    ''' false if left is > than node '''
    if (node.left != None and node.left.data
        return 0

    ''' false if right is < than node '''
    if (node.right != None and node.right.da
        return 0

    ''' false if, recursively, the left or ri
    if (!isBST(node.left) or !isBST(node.rigl
        return 0

    ''' passing all that, it's a BST '''
    return 1

# This code is contributed by Shubham Singh
```

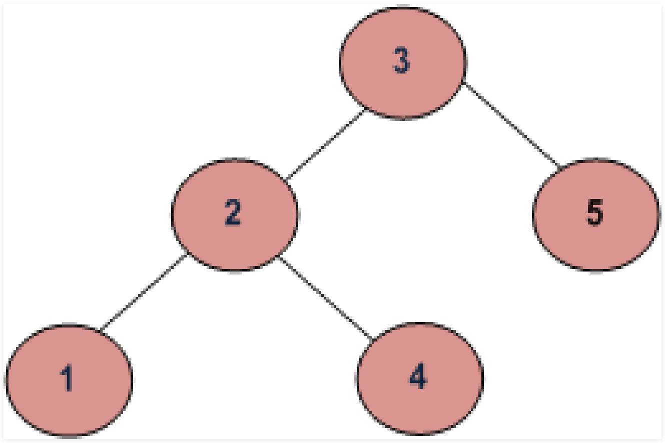
C#

```
bool isBST(Node node)
{
    if (node == null)
        return true;

    /* False if left is > than node */
    if (node.left != null && node.left.data >
        return false;

    /* False if right is < than node */
    if (node.right != null && node.right.data
        return false;

    /* False if, recursively, the left or rig
    if (!isBST(node.left) || !isBST(node.righ
        return false;

    /* Passing all that, it's a BST */
    return true;
}

// This code is contributed by Rajput-Ji
```

Skip to content

```
{
    if (node == null)
        return true;

    /* False if left is > than node */
    if (node.left != null && node.left.data >
        return false;

    /* False if right is < than node */
    if (node.right != null && node.right.data
        return false;

    /* False if, recursively, the left or rig
    if (!isBST(node.left) || !isBST(node.righ
        return false;

    /* Passing all that, it's a BST */
    return true;
}


// This code is contributed by avanitrachhadi

</script>
```

This approach is wrong as this will return true for below binary tree (and below tree is not a BST because 4 is in left subtree of 3)



METHOD 2 (Correct but not efficient)
For each node, check if max value in left subtree is smaller than the node and min value in right subtree greater than the node.

C++

```
/* Returns true if a binary tree is a binary
int isBST(struct node* node)
{
  if (node == NULL)
    return 1;
  if (node->right != NULL && minValue(node->r
    return 0;

  /* false if, recursively, the left or right
  if (!isBST(node->left) || !isBST(node->righ
    return 0;

  /* passing all that, it's a BST */
  return 1;
}

// This code is contributed by shubhamsingh10
```

C

```
/* Returns true if a binary tree is a binary
int isBST(struct node* node)
{
  if (node == NULL)
    return 1;

  /* false if the max of the left is > than u
  if (node->left!=NULL && maxValue(node->left
    return 0;

  /* false if the min of the right is <= than
  if (node->right!=NULL && minValue(node->rig
    return 0;

  /* false if, recursively, the left or right
  if (!isBST(node->left) || !isBST(node->righ
    return 0;

  /* passing all that, it's a BST */
  return 1;
}
```

Java

```
/* Returns true if a binary tree is a binary
int isBST(Node node)
{
  if (node == null)
    return 1;

  /* false if the max of the left is > than u
  if (node.left != null && maxValue(node.left
    return 0;

  /* false if the min of the right is <= than
  if (node.right != null && minValue(node.rig
    return 0;

  /* false if, recursively, the left or right
  if (!isBST(node.left) || !isBST(node.right)
    return 0;

  /* passing all that, it's a BST */
  return 1;
}
```

```
''' Returns true if a binary tree is a binary
def isBST(node):
    if (node == None):
        return 1
    ''' false if the max of the left is > tha
    if (node.left != None and maxValue(node.
        return 0

    ''' false if the min of the right is <= t
    if (node.right != None and minValue(node
        return 0

    ''' false if, recursively, the left or ri
    if (!isBST(node.left) or !isBST(node.righ
        return 0

    ''' passing all that, it's a BST '''
    return 1

# This code is contributed by Shubham Singh
```

C#

```
/* Returns true if a binary tree is a binary
bool isBST(Node node)
{
    if (node == null)
        return true;

    /* false if the max of the left is > than
    if (node.left != null && maxValue(node.le
        return false;

    /* false if the min of the right is <= th
    if (node.right != null && minValue(node.r
        return false;

    /* false if, recursively, the left or rig
    if (!isBST(node.left) || !isBST(node.righ
        return false;

    /* passing all that, it's a BST */
    return true;
}

// This code is contributed by Shubham Singh
```

Javascript

```
<script>

function isBST(node)
{
    if (node == null)
        return true;

    /* False if the max of the left is > than
    if (node.left != null && maxValue(node.le
        return
```

Skip to content

```
    if (!isBST(node.left) || !isBST(node.righ
        return false;

    /* Passing all that, it's a BST */
    return true;
}

// This code is contributed by Shubham Singh

</script>
```

It is assumed that you have helper functions minValue() and maxValue() that return the min or max int value from a non-empty tree

METHOD 3 (Correct and Efficient):
Method 2 above runs slowly since it traverses over some parts of the tree many times. A better solution looks at each node only once. The trick is to write a utility helper function isBSTUtil(struct node* node, int min, int max) that traverses down the tree keeping track of the narrowing min and max allowed values as it goes, looking at each node only once. The initial values for min and max should be INT_MIN and INT_MAX — they narrow from there.

Note: This method is not applicable if there are duplicate elements with value INT_MIN or INT_MAX.

Below is the implementation of the above approach:

C++

```
#include<bits/stdc++.h>

using namespace std;

/* A binary tree node has data,
pointer to left child and
a pointer to right child */
class node
{
    public:
    int data;
    node* left;
    node* right;

    /* Constructor that allocates
    a new node with the given data
    and NULL left and right pointers. */
    node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right = NULL;
    }
};
```

```
int isBST(node* node)
{
    return(isBSTUtil(node, INT_MIN, INT_MAX))
}

/* Returns true if the given
tree is a BST and its values
are >= min and <= max. */
int isBSTUtil(node* node, int min, int max)
{
    /* an empty tree is BST */
    if (node==NULL)
        return 1;

    /* false if this node violates
    the min/max constraint */
    if (node->data < min || node->data > max)
        return 0;

    /* otherwise check the subtrees recursive
    tightening the min or max constraint */
    return
        isBSTUtil(node->left, min, node->data
        isBSTUtil(node->right, node->data+1,
}


/* Driver code*/
int main()
{
    node *root = new node(4);
    root->left = new node(2);
    root->right = new node(5);
    root->left->left = new node(1);
    root->left->right = new node(3);

    if(isBST(root))
        cout<<"Is BST";
    else
        cout<<"Not a BST";

    return 0;
}

// This code is contributed by rathbhupendra
```

C

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

/* A binary tree node has data, pointer to le
   and a pointer to right child */
struct node
{
    int data;
    struct node* left;
    struct node* right;
};

int isBSTUtil(st          le, int min, in
}

/* Returns true if the given tree is a BST ar
   values are >= min and <= max. */
int isBSTUtil(struct node* node, int min, in
{
  /* an empty tree is BST */
  if (node==NULL)
     return 1;

  /* false if this node violates the min/max
  if (node->data < min || node->data > max)
     return 0;

  /* otherwise check the subtrees recursively
   tightening the min or max constraint */
  return
    isBSTUtil(node->left, min, node->data-1)
    isBSTUtil(node->right, node->data+1, max)
}

/* Helper function that allocates a new node
   given data and NULL left and right pointer
struct node* newNode(int data)
{
  struct node* node = (struct node*)
                       malloc(sizeof(struct n
  node->data = data;
  node->left = NULL;
  node->right = NULL;

  return(node);
}

/* Driver program to test above functions*/
int main()
{
  struct node *root = newNode(4);
  root->left        = newNode(2);
  root->right       = newNode(5);
  root->left->left  = newNode(1);
  root->left->right = newNode(3);

  if(isBST(root))
    printf("Is BST");
  else
    printf("Not a BST");

  getchar();
  return 0;
}
```

Java

```
//Java implementation to check if given Binar
//is a BST or not

/* Class containing left and right child of c
 node and key value*/
class Node
{
    int data;
    }
}

public class BinaryTree
{
    //Root of the Binary Tree
    Node root;

    /* can give min and max value according t
    can write a function to find min and max

    /* returns true if given search tree is b
     search tree (efficient version) */
    boolean isBST()  {
        return isBSTUtil(root, Integer.MIN_VA
                               Integer.MAX_VA
    }

    /* Returns true if the given tree is a BS
      values are >= min and <= max. */
    boolean isBSTUtil(Node node, int min, int
    {
        /* an empty tree is BST */
        if (node == null)
            return true;

        /* false if this node violates the mi
        if (node.data < min || node.data > ma
            return false;

        /* otherwise check the subtrees recur
        tightening the min/max constraints */
        // Allow only distinct values
        return (isBSTUtil(node.left, min, nod
                isBSTUtil(node.right, node.da
    }

    /* Driver program to test above functions
    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(4);
        tree.root.left = new Node(2);
        tree.root.right = new Node(5);
        tree.root.left.left = new Node(1);
        tree.root.left.right = new Node(3);

        if (tree.isBST())
            System.out.println("IS BST");
        else
            System.out.println("Not a BST");
    }
}
```

Python3

```
# Python program to check if a binary tree is

INT_MAX = 4294967296
INT_MIN = -4294967296

# A binary tree
```

Skip to content

```
        self.right = None


# Returns true if the given tree is a binary
# (efficient version)
def isBST(node):
    return (isBSTUtil(node, INT_MIN, INT_MAX)

# Retusn true if the given tree is a BST and
# >= min and <= max
def isBSTUtil(node, mini, maxi):

    # An empty tree is BST
    if node is None:
        return True

    # False if this node violates min/max cor
    if node.data < mini or node.data > maxi:
        return False

    # Otherwise check the subtrees recursivel
    # tightening the min or max constraint
    return (isBSTUtil(node.left, mini, node.c
          isBSTUtil(node.right, node.data+1,

# Driver program to test above function
root = Node(4)
root.left = Node(2)
root.right = Node(5)
root.left.left = Node(1)
root.left.right = Node(3)

if (isBST(root)):
    print ("Is BST")
else:
    print ("Not a BST")

# This code is contributed by Nikhil Kumar Si
```

C#

```
using System;

// C# implementation to check if given Binary
//is a BST or not

/* Class containing left and right child of c
 node and key value*/
public class Node
{
    public int data;
    public Node left, right;

    public Node(int item)
    {
        data = item;
        left = right = null;
    }
}

public class BinaryTree
{
```

Skip to content

```
    /* returns true if given search tree is t
     search tree (efficient version) */
    public virtual bool BST
    {
        get
        {
            return isBSTUtil(root, int.MinVal
        }
    }

    /* Returns true if the given tree is a BS
      values are >= min and <= max. */
    public virtual bool isBSTUtil(Node node,
    {
        /* an empty tree is BST */
        if (node == null)
        {
            return true;
        }

        /* false if this node violates the mi
        if (node.data < min || node.data > ma
        {
            return false;
        }

        /* otherwise check the subtrees recur
        tightening the min/max constraints */
        // Allow only distinct values
        return (isBSTUtil(node.left, min, noc
    }

    /* Driver program to test above functions
    public static void Main(string[] args)
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(4);
        tree.root.left = new Node(2);
        tree.root.right = new Node(5);
        tree.root.left.left = new Node(1);
        tree.root.left.right = new Node(3);

        if (tree.BST)
        {
            Console.WriteLine("IS BST");
        }
        else
        {
            Console.WriteLine("Not a BST");
        }
    }
}

  // This code is contributed by Shrikant13
```

Javascript

```
<script>

// Javascript implementation to
// check if given Binary tree
// is a BST or r
```

Skip to content

```
    constructor(item)
    {
        this.data=item;
        this.left=this.right=null;
    }
}

 //Root of the Binary Tree
    let root;

    /* can give min and max value according t
    can write a function to find min and max

    /* returns true if given search tree is b
     search tree (efficient version) */
    function isBST()
    {
        return isBSTUtil(root, Number.MIN_VAL
                               Number.MAX_VAL
    }

    /* Returns true if the given tree is a BS
      values are >= min and <= max. */
    function isBSTUtil(node,min,max)
    {
        /* an empty tree is BST */
        if (node == null)
            return true;

        /* false if this node violates
        the min/max constraints */
        if (node.data < min || node.data > ma
            return false;

        /* otherwise check the subtrees recur
        tightening the min/max constraints */
        // Allow only distinct values
        return (isBSTUtil(node.left, min, nod
                isBSTUtil(node.right, node.da
    }

     /* Driver program to test above function
        root = new Node(4);
        root.left = new Node(2);
        root.right = new Node(5);
        root.left.left = new Node(1);
        root.left.right = new Node(3);

        if (isBST())
            document.write("IS BST<br>");
        else
            document.write("Not a BST<br>");

// This code is contributed by rag2127

</script>
```

Output:

```
IS BST
```

Skip to content

Simplified Method 3

We can simplify method 2 using NULL pointers instead of INT_MIN and INT_MAX values.

C++

```
// C++ program to check if a given tree is BS
#include <bits/stdc++.h>
using namespace std;

/* A binary tree node has data, pointer to
   left child and a pointer to right child */
struct Node
{
    int data;
    struct Node* left, *right;
};

// Returns true if given tree is BST.
bool isBST(Node* root, Node* l=NULL, Node* r=
{
    // Base condition
    if (root == NULL)
        return true;

    // if left node exist then check it has
    // correct data or not i.e. left node's c
    // should be less than root's data
    if (l != NULL and root->data <= l->data)
        return false;

    // if right node exist then check it has
    // correct data or not i.e. right node's
    // should be greater than root's data
    if (r != NULL and root->data >= r->data)
        return false;

    // check recursively for every node.
    return isBST(root->left, l, root) and
           isBST(root->right, root, r);
}

/* Helper function that allocates a new node
   given data and NULL left and right pointer
struct Node* newNode(int data)
{
    struct Node* node = new Node;
    node->data = data;
    node->left = node->right = NULL;
    return (node);
}

/* Driver program to test above functions*/
int main()
{
    struct Node *root = newNode(3);
    root->left        = newNode(2);
    root->right       = newNode(5);
    root->left->left  = newNode(1);
    root->left->right = newNode(4);

    if (isBST(rc
```

Skip to content

Java

```java
// Java program to check if a given tree is B
class Sol
{

// A binary tree node has data, pointer to
//left child && a pointer to right child /
static class Node
{
    int data;
    Node left, right;
};

// Returns true if given tree is BST.
static boolean isBST(Node root, Node l, Node
{
    // Base condition
    if (root == null)
        return true;

    // if left node exist then check it has
    // correct data or not i.e. left node's d
    // should be less than root's data
    if (l != null && root.data <= l.data)
        return false;

    // if right node exist then check it has
    // correct data or not i.e. right node's
    // should be greater than root's data
    if (r != null && root.data >= r.data)
        return false;

    // check recursively for every node.
    return isBST(root.left, l, root) &&
        isBST(root.right, root, r);
}

// Helper function that allocates a new node
//given data && null left && right pointers.
static Node newNode(int data)
{
    Node node = new Node();
    node.data = data;
    node.left = node.right = null;
    return (node);
}

// Driver code
public static void main(String args[])
{
    Node root = newNode(3);
    root.left = newNode(2);
    root.right = newNode(5);
    root.left.left = newNode(1);
    root.left.right = newNode(4);

    if (isBST(root,null,null))
        System.out.print("Is BST");
    else
        System.c [illegible] a BST");
```

Python3

```
""" Program to check if a given Binary
Tree is balanced like a Red-Black Tree """

# Helper function that allocates a new
# node with the given data and None
# left and right poers.
class newNode:

    # Construct to create a new node
    def __init__(self, key):
        self.data = key
        self.left = None
        self.right = None

# Returns true if given tree is BST.
def isBST(root, l = None, r = None):

    # Base condition
    if (root == None) :
        return True

    # if left node exist then check it has
    # correct data or not i.e. left node's da
    # should be less than root's data
    if (l != None and root.data <= l.data) :
        return False

    # if right node exist then check it has
    # correct data or not i.e. right node's d
    # should be greater than root's data
    if (r != None and root.data >= r.data) :
        return False

    # check recursively for every node.
    return isBST(root.left, l, root) and \
        isBST(root.right, root, r)


# Driver Code
if __name__ == '__main__':
    root = newNode(3)
    root.left = newNode(2)
    root.right = newNode(5)
    root.right.left = newNode(1)
    root.right.right = newNode(4)
    #root.right.left.left = newNode(40)
    if (isBST(root,None,None)):
        print("Is BST")
    else:
        print("Not a BST")

# This code is contributed by
# Shubham Singh(SHUBHAMSINGH10)
```

C#

```
// C# program to check if a given tree is BST
using System;
```

Skip to content

```
{
    public int data;
    public Node left, right;
};

// Returns true if given tree is BST.
static Boolean isBST(Node root, Node l, Node
{
    // Base condition
    if (root == null)
        return true;

    // if left node exist then check it has
    // correct data or not i.e. left node's c
    // should be less than root's data
    if (l != null && root.data <= l.data)
        return false;

    // if right node exist then check it has
    // correct data or not i.e. right node's
    // should be greater than root's data
    if (r != null && root.data >= r.data)
        return false;

    // check recursively for every node.
    return isBST(root.left, l, root) &&
        isBST(root.right, root, r);
}

// Helper function that allocates a new node
//given data && null left && right pointers.
static Node newNode(int data)
{
    Node node = new Node();
    node.data = data;
    node.left = node.right = null;
    return (node);
}

// Driver code
public static void Main(String []args)
{
    Node root = newNode(3);
    root.left = newNode(2);
    root.right = newNode(5);
    root.left.left = newNode(1);
    root.left.right = newNode(4);

    if (isBST(root,null,null))
        Console.Write("Is BST");
    else
        Console.Write("Not a BST");
}
}

// This code is contributed by 29AjayKumar
```

Javascript

```
<script>

    // JavaScrip           heck if a giver
```

Skip to content

```
            this.data = data;
        }
    }

    // Returns true if given tree is BST.
    function isBST(root, l, r)
    {
        // Base condition
        if (root == null)
            return true;

        // if left node exist then check it h
        // correct data or not i.e. left node
        // should be less than root's data
        if (l != null && root.data <= l.data
            return false;

        // if right node exist then check it
        // correct data or not i.e. right noc
        // should be greater than root's data
        if (r != null && root.data >= r.data
            return false;

        // check recursively for every node.
        return isBST(root.left, l, root) &&
            isBST(root.right, root, r);
    }

    // Helper function that allocates a new r
    //given data && null left && right pointe
    function newNode(data)
    {
        let node = new Node(data);
        return (node);
    }

    let root = newNode(3);
    root.left = newNode(2);
    root.right = newNode(5);
    root.left.left = newNode(1);
    root.left.right = newNode(4);

    if (isBST(root,null,null))
        document.write("Is BST");
    else
        document.write("Not a BST");

</script>
```

Output:

```
Not a BST
```

Thanks to **Abhinesh Garhwal** for suggesting above solution.

METHOD 4(Using In-Order Traversal)

Thanks to *LJW489* for suggesting this method.

3) Check if the temp array is sorted in ascending order, if it is, then the tree is BST.

Time Complexity: O(n)

We can avoid the use of a Auxiliary Array. While doing In-Order traversal, we can keep track of previously visited node. If the value of the currently visited node is less than the previous value, then tree is not BST. Thanks to *ygos* for this space optimization.

C++

```
bool isBST(node* root)
{
    static node *prev = NULL;

    // traverse the tree in inorder fashion
    // and keep track of prev node
    if (root)
    {
        if (!isBST(root->left))
        return false;

        // Allows only distinct valued nodes
        if (prev != NULL &&
            root->data <= prev->data)
        return false;

        prev = root;

        return isBST(root->right);
    }

    return true;
}

// This code is contributed by rathbhupendra
```

C

```
bool isBST(struct node* root)
{
    static struct node *prev = NULL;

    // traverse the tree in inorder fashion a
    if (root)
    {
        if (!isBST(root->left))
          return false;

        // Allows only distinct valued nodes
        if (prev != NULL && root->data <= pre
          return false;

        prev = root;

        return isBST(root->right);
    }
```

Skip to content

Java

```java
// Java implementation to check if given Bina
// is a BST or not

/* Class containing left and right child of c
 node and key value*/
class Node
{
    int data;
    Node left, right;

    public Node(int item)
    {
        data = item;
        left = right = null;
    }
}

public class BinaryTree
{
    // Root of the Binary Tree
    Node root;

    // To keep tract of previous node in Inor
    Node prev;

    boolean isBST()  {
        prev = null;
        return isBST(root);
    }

    /* Returns true if given search tree is b
       search tree (efficient version) */
    boolean isBST(Node node)
    {
        // traverse the tree in inorder fashi
        // keep a track of previous node
        if (node != null)
        {
            if (!isBST(node.left))
                return false;

            // allows only distinct values nc
            if (prev != null && node.data <=
                return false;
            prev = node;
            return isBST(node.right);
        }
        return true;
    }

    /* Driver program to test above functions
    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(4);
        tree.root.left = new Node(2);
        tree.root.right = new Node(5);
        tree.root.left.left = new Node(1);
        tree.root.left.right = new Node(3);

        if (tree.isBST())
            Syst       ("IS BST");
```

Skip to content

Python3

```
# Python implementation to check if
# given Binary tree is a BST or not

# A binary tree node containing data
# field, left and right pointers
class Node:
    # constructor to create new node
    def __init__(self, val):
        self.data = val
        self.left = None
        self.right = None

# global variable prev - to keep track
# of previous node during Inorder
# traversal
prev = None

# function to check if given binary
# tree is BST
def isbst(root):

    # prev is a global variable
    global prev
    prev = None
    return isbst_rec(root)


# Helper function to test if binary
# tree is BST
# Traverse the tree in inorder fashion
# and keep track of previous node
# return true if tree is Binary
# search tree otherwise false
def isbst_rec(root):

    # prev is a global variable
    global prev

    # if tree is empty return true
    if root is None:
        return True

    if isbst_rec(root.left) is False:
        return False

    # if previous node'data is found
    # greater than the current node's
    # data return false
    if prev is not None and prev.data > root
        return False

    # store the current node in prev
    prev = root
    return isbst_rec(root.right)


# driver code to test above function
root = Node(4)
root.left = Node(2)
root.right = Node(5)
root.left.left = Node(1)
```

Skip to content

```
# This code is contributed by
# Shweta Singh(shweta44)
```

C#

```
// C# implementation to check if
// given Binary tree is a BST or not
using System;

/* Class containing left and
right child of current node
and key value*/
class Node
{
    public int data;
    public Node left, right;

    public Node(int item)
    {
        data = item;
        left = right = null;
    }
}

public class BinaryTree
{
    // Root of the Binary Tree
    Node root;

    // To keep tract of previous node
    // in Inorder Traversal
    Node prev;

    Boolean isBST()
    {
        prev = null;
        return isBST(root);
    }

    /* Returns true if given search tree is t
    search tree (efficient version) */
    Boolean isBST(Node node)
    {
        // traverse the tree in inorder fashi
        // keep a track of previous node
        if (node != null)
        {
            if (!isBST(node.left))
                return false;

            // allows only distinct values nc
            if (prev != null &&
                node.data <= prev.data )
                return false;
            prev = node;
            return isBST(node.right);
        }
        return true;
    }

    // Driver Cc
```

```
        tree.root.left.left = new Node(1);
        tree.root.left.right = new Node(3);

        if (tree.isBST())
            Console.WriteLine("IS BST");
        else
            Console.WriteLine("Not a BST");
    }
}

// This code is contributed by Rajput-Ji
```

Javascript

```
<script>
// Javascript implementation to check if give
// is a BST or not

/* Class containing left and right child of c
 node and key value*/
class Node
{
    constructor(item)
    {
        this.data = item;
        this.left = this.right=null;
    }
}

// Root of the Binary Tree
let root;

// To keep tract of previous node in Inorder
let prev;

function isBST()
{
    prev = null;
    return _isBST(root);
}

/* Returns true if given search tree is binar
       search tree (efficient version) */
function _isBST(node)
{

    // traverse the tree in inorder fashion a
        // keep a track of previous node
        if (node != null)
        {
            if (!_isBST(node.left))
                return false;

            // allows only distinct values nc
            if (prev != null && node.data <=
                return false;
            prev = node;
            return _isBST(node.right);
        }
        return true;
}
```

Skip to content

```
if (isBST())
    document.write("IS BST");
else
    document.write("Not a BST");

// This code is contributed by unknown2108
</script>
```

The use of a static variable can also be avoided by using a reference to the prev node as a parameter.

C++

```
// C++ program to check if a given tree is BS
#include <bits/stdc++.h>
using namespace std;

/* A binary tree node has data, pointer to
left child and a pointer to right child */
struct Node
{
    int data;
    struct Node* left, *right;

    Node(int data)
    {
        this->data = data;
        left = right = NULL;
    }
};


bool isBSTUtil(struct Node* root, Node *&pre
{
    // traverse the tree in inorder fashion a
    // keep track of prev node
    if (root)
    {
        if (!isBSTUtil(root->left, prev))
          return false;

        // Allows only distinct valued nodes
        if (prev != NULL && root->data <= pre
          return false;

        prev = root;

        return isBSTUtil(root->right, prev);
    }

    return true;
}

bool isBST(Node *root)
{
   Node *prev = NULL;
   return isBSTUtil(root, prev);
}

/* Driver progra            e functions*/
```

Skip to content

```
    root->left->right = new Node(4);

    if (isBST(root))
        cout << "Is BST";
    else
        cout << "Not a BST";

    return 0;
}
```

Java

```
// Java program to check if a given tree is E
import java.io.*;

class GFG {
    /* A binary tree node has data, pointer t
    left child and a pointer to right child *
    public static class Node
    {
        public int data;
        public Node left, right;

        public Node(int data)
        {
            this.data = data;
            left = right = null;
        }
    };

    static  Node prev;

    static Boolean isBSTUtil(Node root)
    {
        // traverse the tree in inorder fashi
        // keep track of prev node
        if (root != null)
        {
            if (!isBSTUtil(root.left))
            return false;

            // Allows only distinct valued nc
            if (prev != null &&
                root.data <= prev.data)
            return false;

            prev = root;

            return isBSTUtil(root.right);
        }
        return true;
    }

    static Boolean isBST(Node root)
    {
        return isBSTUtil(root);
    }

    // Driver Code
    public static void main (String[] args)
    {
        Node roc          );
```

```
            System.out.println("Is BST");
        else
            System.out.println("Not a BST");
    }
}

// This code is contributed by Shubham Singh
```

Python3

```
# Python3 program to check
# if a given tree is BST.
import math

# A binary tree node has data,
# pointer to left child and
# a pointer to right child
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

def isBSTUtil(root, prev):

    # traverse the tree in inorder fashion
    # and keep track of prev node
    if (root != None):
        if (isBSTUtil(root.left, prev) == Tru
            return False

        # Allows only distinct valued nodes
        if (prev != None and
            root.data <= prev.data):
            return False

        prev = root
        return isBSTUtil(root.right, prev)

    return True

def isBST(root):
    prev = None
    return isBSTUtil(root, prev)

# Driver Code
if __name__ == '__main__':
    root = Node(3)
    root.left = Node(2)
    root.right = Node(5)
    root.right.left = Node(1)
    root.right.right = Node(4)
    #root.right.left.left = Node(40)

    if (isBST(root) == None):
        print("Is BST")
    else:
        print("Not a BST")

# This code is contributed by Srathore
```

Skip to content

```
{
/* A binary tree node has data, pointer to
left child and a pointer to right child */
public class Node
{
    public int data;
    public Node left, right;

    public Node(int data)
    {
        this.data = data;
        left = right = null;
    }
};

static Node prev;

static Boolean isBSTUtil(Node root)
{
    // traverse the tree in inorder fashion a
    // keep track of prev node
    if (root != null)
    {
        if (!isBSTUtil(root.left))
        return false;

        // Allows only distinct valued nodes
        if (prev != null &&
            root.data <= prev.data)
        return false;

        prev = root;

        return isBSTUtil(root.right);
    }
    return true;
}

static Boolean isBST(Node root)
{
    return isBSTUtil(root);
}

// Driver Code
public static void Main(String[] args)
{
    Node root = new Node(3);
    root.left = new Node(2);
    root.right = new Node(5);
    root.left.left = new Node(1);
    root.left.right = new Node(4);

    if (isBST(root))
        Console.WriteLine("Is BST");
    else
        Console.WriteLine("Not a BST");
}
}

// This code is contributed by Rajput-Ji
```

Javascript

Skip to content

```
            this.left = null;
            this.right = null;
            this.data = data;
        }
    }

    let prev;

    function isBSTUtil(root)
    {

        // traverse the tree in inorder fashi
        // keep track of prev node
        if (root != null)
        {
            if (!isBSTUtil(root.left))
                return false;

            // Allows only distinct valued nc
            if (prev != null && root.data <=
                return false;

            prev = root;

            return isBSTUtil(root.right);
        }
        return true;
    }

    function isBST(root)
    {
        return isBSTUtil(root);
    }

    let root = new Node(3);
    root.left = new Node(2);
    root.right = new Node(5);
    root.left.left = new Node(1);
    root.left.right = new Node(4);

    if (isBST(root))
        document.write("Is BST");
    else
        document.write("Not a BST");

// This code is contributed by divyeshrabadiy
</script>
```

Output:

```
Not a BST
```