Binary Search Tree

[**Data Structure and Algorithms Course**](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=gfg&utm_medium=header+link+click&utm_campaign=dsa+course+tracker&utm_term=dsa+course+promo&utm_content=bst-lp)

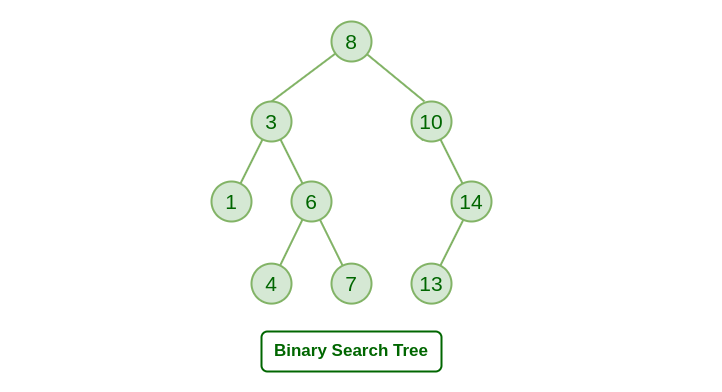
[**Practice Problems on Binary Search Tree !**](https://practice.geeksforgeeks.org/explore/?category%5B%5D=Binary%20Search%20Tree&page=1&category%5B%5D=Binary%20Search%20Tree&utm_source=gfg&utm_medium=header+link+click&utm_campaign=practice+tracker&utm_term=practice+promo&utm_content=bst-lp)

[**Recent Articles on Binary Search Tree !**](https://www.geeksforgeeks.org/category/binary-search-tree/?utm_source=gfg&utm_medium=header+link+click&utm_campaign=recent+article+tracker&utm_term=recent+article+tracker&utm_content=bst-lp)

**What is Binary Search Tree?**

**Binary Search Tree** is a node-based binary tree data structure which has the following properties:

* The left subtree of a node contains only nodes with keys lesser than 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.



*Binary Search Tree*

**Topic:**

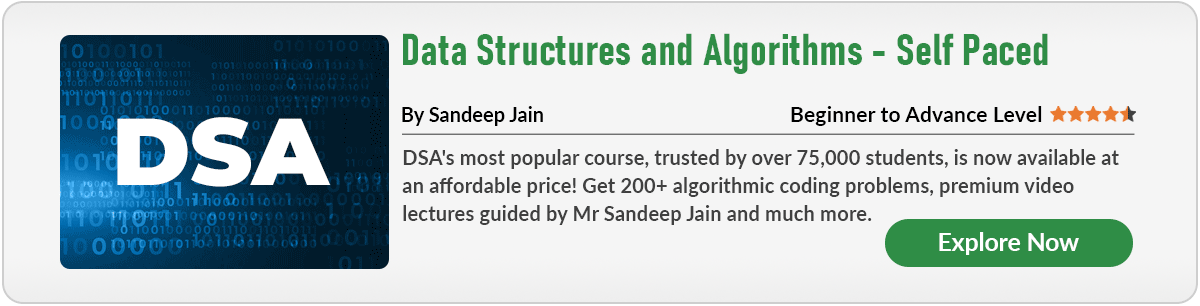
* [Introduction](https://www.geeksforgeeks.org/binary-search-tree-data-structure/?ref=ghm#introduction)
* [Basic Operations](https://www.geeksforgeeks.org/binary-search-tree-data-structure/?ref=ghm#basic)
* [Standard problem on BST](https://www.geeksforgeeks.org/binary-search-tree-data-structure/?ref=ghm#standard)
* [Quick Links](https://www.geeksforgeeks.org/binary-search-tree-data-structure/?ref=ghm#quick)

**Introduction:**

1. [Introduction to Binary Search Tree – Data Structure and Algorithm Tutorials](https://www.geeksforgeeks.org/introduction-to-binary-search-tree-data-structure-and-algorithm-tutorials/)
2. [Applications of BST](https://www.geeksforgeeks.org/applications-of-bst/)
3. [Application, Advantages and Disadvantages of Binary Search Tree](https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-binary-search-tree/)

**Basic Operations:**

1. [Insertion in Binary Search Tree](https://www.geeksforgeeks.org/insertion-in-binary-search-tree/)
2. [Searching in Binary Search Tree](https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/)
3. [Deletion in Binary Search Tree](https://www.geeksforgeeks.org/binary-search-tree-set-2-delete/)
4. [Binary Search Tree (BST) Traversals – Inorder, Preorder, Post Order](https://www.geeksforgeeks.org/binary-search-tree-traversal-inorder-preorder-post-order/)
5. [Convert a normal BST to Balanced BST](https://www.geeksforgeeks.org/convert-normal-bst-balanced-bst/)

[](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=page&utm_medium=page&utm_campaign=dsa-self-paced)

**Standard problems on BST**

* **Easy:**
  1. [Iterative searching in Binary Search Tree](https://www.geeksforgeeks.org/iterative-searching-binary-search-tree/)
  2. [A program to check if a binary tree is BST or not](https://www.geeksforgeeks.org/a-program-to-check-if-a-binary-tree-is-bst-or-not/)
  3. [Binary Tree to Binary Search Tree Conversion](https://www.geeksforgeeks.org/binary-tree-to-binary-search-tree-conversion/)
  4. [Find the node with minimum value in a Binary Search Tree](https://www.geeksforgeeks.org/find-the-minimum-element-in-a-binary-search-tree/)
  5. [Check if an array represents Inorder of Binary Search tree or not](https://www.geeksforgeeks.org/check-array-represents-inorder-binary-search-tree-not/)
  6. [How to determine if a binary tree is height-balanced?](https://www.geeksforgeeks.org/how-to-determine-if-a-binary-tree-is-balanced/)
  7. [Sorted Array to Balanced BST](https://www.geeksforgeeks.org/sorted-array-to-balanced-bst/)
  8. [Check for Identical BSTs without building the trees](https://www.geeksforgeeks.org/check-for-identical-bsts-without-building-the-trees/)
  9. [Convert a normal BST to Balanced BST](https://www.geeksforgeeks.org/convert-normal-bst-balanced-bst/)
  10. [Convert BST to Min Heap](https://www.geeksforgeeks.org/convert-bst-min-heap/)
  11. [Second largest element in BST](https://www.geeksforgeeks.org/second-largest-element-in-binary-search-tree-bst/)
  12. [Add all greater values to every node in a given BST](https://www.geeksforgeeks.org/add-greater-values-every-node-given-bst/)
  13. [Check if two BSTs contain same set of elements](https://www.geeksforgeeks.org/check-two-bsts-contain-set-elements/)
  14. [Sum of k smallest elements in BST](https://www.geeksforgeeks.org/sum-k-smallest-elements-bst/)
* **Medium:**
  1. [Construct BST from given preorder traversal | Set 1](https://www.geeksforgeeks.org/construct-bst-from-given-preorder-traversa/)
  2. [Sorted Linked List to Balanced BST](https://www.geeksforgeeks.org/sorted-linked-list-to-balanced-bst/)
  3. [Transform a BST to greater sum tree](https://www.geeksforgeeks.org/transform-bst-sum-tree/)
  4. [BST to a Tree with sum of all smaller keys](https://www.geeksforgeeks.org/bst-tree-sum-smaller-keys/)
  5. [Construct BST from its given level order traversal](https://www.geeksforgeeks.org/construct-bst-given-level-order-traversal/)
  6. [Check if the given array can represent Level Order Traversal of Binary Search Tree](https://www.geeksforgeeks.org/check-given-array-can-represent-level-order-traversal-binary-search-tree/)
  7. [Lowest Common Ancestor in a Binary Search Tree](https://www.geeksforgeeks.org/lowest-common-ancestor-in-a-binary-search-tree/)
  8. [Find k-th smallest element in BST (Order Statistics in BST)](https://www.geeksforgeeks.org/find-k-th-smallest-element-in-bst-order-statistics-in-bst/)
  9. [K’th Largest element in BST using constant extra space](https://www.geeksforgeeks.org/kth-largest-element-bst-using-constant-extra-space/)
  10. [Largest number in BST which is less than or equal to N](https://www.geeksforgeeks.org/largest-number-bst-less-equal-n/)
  11. [Find distance between two nodes of a Binary Search Tree](https://www.geeksforgeeks.org/find-distance-two-nodes-binary-search-tree/)
  12. [Largest BST in a Binary Tree | Set 2](https://www.geeksforgeeks.org/largest-bst-binary-tree-set-2/)
  13. [Remove all leaf nodes from the binary search tree](https://www.geeksforgeeks.org/remove-leaf-nodes-binary-search-tree/)
  14. [Inorder Successor in Binary Search Tree](https://www.geeksforgeeks.org/inorder-successor-in-binary-search-tree/)
  15. [Find a pair with given sum in BST](https://www.geeksforgeeks.org/find-pair-given-sum-bst/)
  16. [Maximum element between two nodes of BST](https://www.geeksforgeeks.org/maximum-element-two-nodes-bst/)
  17. [Find the largest BST subtree in a given Binary Tree](https://www.geeksforgeeks.org/find-the-largest-subtree-in-a-tree-that-is-also-a-bst/)
  18. [Find a pair with given sum in a Balanced BST](https://www.geeksforgeeks.org/find-a-pair-with-given-sum-in-bst/)
  19. [Two nodes of a BST are swapped, correct the BST](https://www.geeksforgeeks.org/fix-two-swapped-nodes-of-bst/)
  20. [How to handle duplicates in Binary Search Tree?](https://www.geeksforgeeks.org/how-to-handle-duplicates-in-binary-search-tree/)
  21. [Leaf nodes from Preorder of a Binary Search Tree (Using Recursion)](https://www.geeksforgeeks.org/leaf-nodes-preorder-binary-search-treeusing-recursion/)
* **Hard:**
  1. [Construct all possible BSTs for keys 1 to N](https://www.geeksforgeeks.org/construct-all-possible-bsts-for-keys-1-to-n/)
  2. [In-place Convert BST into a Min-Heap](https://www.geeksforgeeks.org/in-place-convert-bst-into-a-min-heap/)
  3. [Check given array of size n can represent BST of n levels or not](https://www.geeksforgeeks.org/check-given-array-of-size-n-can-represent-bst-of-n-levels-or-not/)
  4. [Merge two BSTs with limited extra space](https://www.geeksforgeeks.org/merge-two-bsts-with-limited-extra-space/)
  5. [K’th Largest Element in BST when modification to BST is not allowed](https://www.geeksforgeeks.org/kth-largest-element-in-bst-when-modification-to-bst-is-not-allowed/)
  6. [Check if given sorted sub-sequence exists in binary search tree](https://www.geeksforgeeks.org/check-if-given-sorted-sub-sequence-exists-in-binary-search-tree/)
  7. [Maximum Unique Element in every subarray of size K](https://www.geeksforgeeks.org/maximum-unique-element-every-subarray-size-k/)
  8. [Count pairs from two BSTs whose sum is equal to a given value x](https://www.geeksforgeeks.org/count-pairs-from-two-bsts-whose-sum-is-equal-to-a-given-value-x/)
  9. [Print BST keys in given Range | O(1) Space](https://www.geeksforgeeks.org/print-bst-keys-in-given-range-o1-space/)
  10. [Inorder predecessor and successor for a given key in BST](https://www.geeksforgeeks.org/inorder-predecessor-successor-given-key-bst/)
  11. [Find if there is a triplet in a Balanced BST that adds to zero](https://www.geeksforgeeks.org/find-if-there-is-a-triplet-in-bst-that-adds-to-0/)
  12. [Replace every element with the least greater element on its right](https://www.geeksforgeeks.org/replace-every-element-with-the-least-greater-element-on-its-right/)
  13. [Count inversions in an array | Set 2 (Using Self-Balancing BST)](https://www.geeksforgeeks.org/count-inversions-in-an-array-set-2-using-self-balancing-bst/)
  14. [Leaf nodes from Preorder of a Binary Search Tree](https://www.geeksforgeeks.org/leaf-nodes-preorder-binary-search-tree/)
  15. [Minimum Possible value of |ai + aj – k| for given array and k.](https://www.geeksforgeeks.org/minimum-possible-value-ai-aj-k-given-array-k/)
  16. [Special two digit numbers in a Binary Search Tree](https://www.geeksforgeeks.org/special-two-digit-numbers-in-a-binary-search-tree/)
  17. [Merge Two Balanced Binary Search Trees](https://www.geeksforgeeks.org/merge-two-balanced-binary-search-trees/)

**Quick Links :**

* [‘Practice Problems’ on Binary Search Tree](https://practice.geeksforgeeks.org/topics/Binary%20Search%20Tree/?ref=taocp)
* [‘Quizzes’ on Binary Search Tree](https://www.geeksforgeeks.org/data-structure-gq/binary-search-trees-gq/)
* [‘Quizzes’ on Balanced Binary Search Trees](https://www.geeksforgeeks.org/data-structure-gq/balanced-binary-search-trees-gq/)
* [Videos](https://www.youtube.com/channel/UC0RhatS1pyxInC00YKjjBqQ/search?query=binary+search+tree)

*From <*[*https://www.geeksforgeeks.org/binary-search-tree-data-structure/?ref=ghm*](https://www.geeksforgeeks.org/binary-search-tree-data-structure/?ref=ghm)*>*

**Easy Questions:**

**Iterative searching in Binary Search Tree**

* Difficulty Level : [Basic](https://www.geeksforgeeks.org/basic/)
* Last Updated : 01 Dec, 2022
* Read
* Discuss(2)
* Courses
* Practice
* Video

Given a [binary search tree](https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/) and a key. Check the given key exists in BST or not without recursion.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

Please refer [binary search tree insertion](https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/) for recursive search.

* C++
* Java
* Python3
* C#
* Javascript

# Python program to demonstrate searching operation

# in binary search tree without recursion

**class** newNode:

    # Constructor to create a new node

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.left **=** None

        self.right **=** None

# Function to check the given

# key exist or not

**def** iterativeSearch(root, key):

    # Traverse until root reaches

    # to dead end

**while** root !**=** None:

        # pass right subtree as new tree

**if** key > root.data:

            root **=** root.right

        # pass left subtree as new tree

**elif** key < root.data:

            root **=** root.left

**else**:

**return** True # if the key is found return 1

**return** False

# A utility function to insert a

# new Node with given key in BST

**def** insert(Node, data):

    # If the tree is empty, return

    # a new Node

**if** Node **==** None:

**return** newNode(data)

    # Otherwise, recur down the tree

**if** data < Node.data:

        Node.left **=** insert(Node.left, data)

**elif** data > Node.data:

        Node.right **=** insert(Node.right, data)

    # return the (unchanged) Node pointer

**return** Node

# Driver Code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    # Let us create following BST

    # 50

    # 30     70

    # / \ / \

    # 20 40 60 80

    root **=** None

    root **=** insert(root, 50)

    insert(root, 30)

    insert(root, 20)

    insert(root, 40)

    insert(root, 70)

    insert(root, 60)

    insert(root, 80)

**if** iterativeSearch(root, 15):

        print("Yes")

**else**:

**print**("No")

# This code is contributed by PranchalK

**Output**

No

**Time Complexity:**O(h), here h is the height of the BST.

**Auxiliary Space:**O(1), as constant extra space is used.

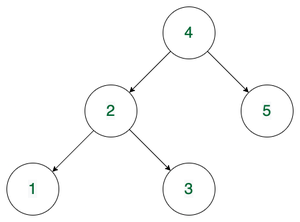
*From <*[*https://www.geeksforgeeks.org/iterative-searching-binary-search-tree/*](https://www.geeksforgeeks.org/iterative-searching-binary-search-tree/)*>*

**A program to check if a Binary Tree is BST or not**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 15 Dec, 2022
* Read
* Discuss(603)
* Courses
* Practice
* Video

*A binary search tree (BST) is a node-based binary tree data structure that 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.*
* *Each node (item in the tree) has a distinct key.*



*BST*

Recommended Problem

Check for BST

[Binary Search Tree](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Binary%20Search%20Tree&sortBy=submissions)

[Tree](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Tree&sortBy=submissions)

+1 more

[Accolite](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Accolite&sortBy=submissions)

[Adobe](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Adobe&sortBy=submissions)

+19 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/check-for-bst/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 4L

**Naive Approach:**

*The idea is to 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.*

Follow the below steps to solve the problem:

* If the current node is null then return true
* If the value of the left child of the node is greater than or equal to the current node then return false
* If the value of the right child of the node is less than or equal to the current node then return false
* If the left subtree or the right subtree is not a BST then return false
* Else return true

*From <*[*https://www.geeksforgeeks.org/a-program-to-check-if-a-binary-tree-is-bst-or-not/*](https://www.geeksforgeeks.org/a-program-to-check-if-a-binary-tree-is-bst-or-not/)*>*

**Approach (Efficient):**

*The idea 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 the value INT\_MIN or INT\_MAX.

Follow the below steps to solve the problem:

* Call the isBstUtil function for the root node and set the minimum value as INT\_MIN and the maximum value as INT\_MAX
* If the current node is NULL then return true
* If the value of the node is less than the minimum value possible or greater than the maximum value possible then return false
* Call the same function for the left and the right subtree and narrow down the minimum and maximum values for these calls accordingly

Below is the implementation of the above approach:

* C++
* C
* Java
* Python3
* C#
* Javascript

# Python program to check if a binary tree is bst or not

INT\_MAX **=** 4294967296

INT\_MIN **= -**4294967296

# A binary tree node

**class** Node:

    # Constructor to create a new node

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.left **=** None

        self.right **=** None

# Returns true if the given tree is a binary search tree

# (efficient version)

**def** isBST(node):

**return** (isBSTUtil(node, INT\_MIN, INT\_MAX))

# Returns true if the given tree is a BST and its values

# >= 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 constraint

**if** node.data < mini **or** node.data > maxi:

**return** False

    # Otherwise check the subtrees recursively

    # tightening the min or max constraint

**return** (isBSTUtil(node.left, mini, node.data **-** 1) **and**

            isBSTUtil(node.right, node.data**+**1, maxi))

# Driver code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

  root **=** Node(4)

  root.left **=** Node(2)

  root.right **=** Node(5)

  root.left.left **=** Node(1)

  root.left.right **=** Node(3)

  # Function call

**if** (isBST(root)):

**print**("Is BST")

**else**:

      print("Not a BST")

# This code is contributed by Nikhil Kumar Singh(nickzuck\_007)

**Output**

Is BST

**Time Complexity:** O(N), Where N is the number of nodes in the tree

**Auxiliary Space:** O(1), if Function Call Stack size is not considered, otherwise O(H) where H is the height of the tree

**Check whether the binary tree is BST or not using**[**inorder traversal**](https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/)**:**

*The idea is to use Inorder traversal of a binary search tree generates output, sorted in ascending order. So generate inorder traversal of the  given binary tree and check if the values are sorted or not*

Follow the below steps to solve the problem:

* Do In-Order Traversal of the given tree and store the result in a temp array.
* This method assumes that there are no duplicate values in the tree
* Check if the temp array is sorted in ascending order, if it is, then the tree is BST.

**Note:** We can avoid the use of an Auxiliary Array. While doing In-Order traversal, we can keep track of previously visited nodes. If the value of the currently visited node is less than the previous value, then the tree is not BST.

Below is the implementation of the above approach:

* C++
* Java
* Python3
* C#
* Javascript

# 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) **==** True):

**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)

    # Function call

**if** (isBST(root) **==** None):

**print**("Is BST")

**else**:

**print**("Not a BST")

# This code is contributed by Srathore

**Output**

Not a BST

**Time Complexity:**O(N), Where N is the number of nodes in the tree

**Auxiliary Space:** O(H), Here H is the height of the tree and the extra space is used due to the function call stack.

*From <*[*https://www.geeksforgeeks.org/a-program-to-check-if-a-binary-tree-is-bst-or-not/*](https://www.geeksforgeeks.org/a-program-to-check-if-a-binary-tree-is-bst-or-not/)*>*

**Binary Tree to Binary Search Tree Conversion**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 17 Jun, 2022
* Read
* Discuss(93)
* Courses
* Practice
* Video

Given a Binary Tree, convert it to a Binary Search Tree. The conversion must be done in such a way that keeps the original structure of Binary Tree.

**Examples**

Example 1  
Input:  
 10  
 / \  
 2 7  
 / \  
 8 4  
Output:  
 8  
 / \  
 4 10  
 / \  
 2 7

Example 2  
Input:  
 10  
 / \  
 30 15  
 / \  
 20 5  
Output:  
 15  
 / \  
 10 20  
 / \  
 5 30

***Solution:***

*Following is a 3 step solution for converting Binary tree to Binary Search Tree.*

1. *Create a temp array arr[] that stores inorder traversal of the tree. This step takes O(n) time.*
2. *Sort the temp array arr[]. Time complexity of this step depends upon the sorting algorithm. In the following implementation, Quick Sort is used which takes (n^2) time. This can be done in O(nLogn) time using Heap Sort or Merge Sort.*
3. *Again do inorder traversal of tree and copy array elements to tree nodes one by one. This step takes O(n) time.*

*Following is the implementation of the above approach. The main function to convert is highlighted in the following code.*

**Implementation:**

* C++
* C
* Java
* Python3
* C#
* Javascript

# Program to convert binary tree to BST

# A binary tree node

**class** Node:

    # Constructor to create a new node

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.left **=** None

        self.right **=** None

# Helper function to store the inorder traversal of a tree

**def** storeInorder(root, inorder):

    # Base Case

**if** root **is** None:

**return**

    # First store the left subtree

    storeInorder(root.left, inorder)

    # Copy the root's data

    inorder.append(root.data)

    # Finally store the right subtree

    storeInorder(root.right, inorder)

# A helper function to count nodes in a binary tree

**def** countNodes(root):

**if** root **is** None:

**return** 0

**return** countNodes(root.left) **+** countNodes(root.right) **+** 1

# Helper function that copies contents of sorted array

# to Binary tree

**def** arrayToBST(arr, root):

    # Base Case

**if** root **is** None:

**return**

    # First update the left subtree

    arrayToBST(arr, root.left)

    # now update root's data delete the value from array

    root.data **=** arr[0]

    arr.pop(0)

    # Finally update the right subtree

    arrayToBST(arr, root.right)

# This function converts a given binary tree to BST

**def** binaryTreeToBST(root):

    # Base Case: Tree is empty

**if** root **is** None:

**return**

    # Count the number of nodes in Binary Tree so that

    # we know the size of temporary array to be created

    n **=** countNodes(root)

    # Create the temp array and store the inorder traversal

    # of tree

    arr **=** []

    storeInorder(root, arr)

    # Sort the array

    arr.sort()

    # copy array elements back to binary tree

    arrayToBST(arr, root)

# Print the inorder traversal of the tree

**def** printInorder(root):

**if** root **is** None:

**return**

    printInorder(root.left)

    print (root.data,end**=**" ")

    printInorder(root.right)

# Driver program to test above function

root **=** Node(10)

root.left **=** Node(30)

root.right **=** Node(15)

root.left.left **=** Node(20)

root.right.right **=** Node(5)

# Convert binary tree to BST

binaryTreeToBST(root)

**print** ("Following is the inorder traversal of the converted BST")

printInorder(root)

# This code is contributed by Nikhil Kumar Singh(nickzuck\_007)

**Output**

Following is the inorder traversal of the converted BST  
5 10 15 20 30

**Complexity Analysis:**

* **Time Complexity:** O(nlogn). This is the complexity of the sorting algorithm which we are using after first in-order traversal, rest of the operations take place in linear time.
* **Auxiliary Space:** O(n). Use of data structure ‘array’ to store in-order traversal.

*From <*[*https://www.geeksforgeeks.org/binary-tree-to-binary-search-tree-conversion/*](https://www.geeksforgeeks.org/binary-tree-to-binary-search-tree-conversion/)*>*

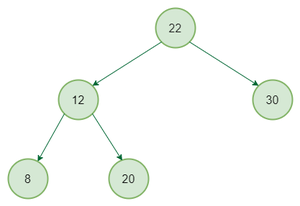
**Find the node with minimum value in a Binary Search Tree**

* Difficulty Level : [Basic](https://www.geeksforgeeks.org/basic/)
* Last Updated : 12 Dec, 2022
* Read
* Discuss(82)
* Courses
* Practice
* Video

Write a function to find the node with minimum value in a Binary Search Tree.

**Example:**

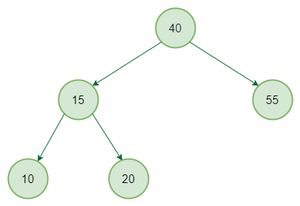
***Input:***



*first example BST*

***Output:****8*

***Input:***



*second example BST*

***Output:****10*

Recommended Problem

Minimum element in BST

[Binary Search Tree](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Binary%20Search%20Tree&sortBy=submissions)

[Tree](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Tree&sortBy=submissions)

+1 more

[Microsoft](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Microsoft&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/minimum-element-in-bst/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 99.4K

**Approach:** To solve the problem follow the below idea:

*This is quite simple. Just traverse the node from root to left recursively until left is NULL. The node whose left is NULL is the node with minimum value*

Below is the implementation of the above approach:

* C++
* C
* Java
* Python3
* C#
* PHP
* Javascript

# Python3 program to find the node with minimum value in bst

# A binary tree node

**class** Node:

    # Constructor to create a new node

**def** \_\_init\_\_(self, key):

        self.data **=** key

        self.left **=** None

        self.right **=** None

""" Give a binary search tree and a number,

inserts a new node with the given number in

the correct place in the tree. Returns the new

root pointer which the caller should then use

(the standard trick to avoid using reference

parameters). """

**def** insert(node, data):

    # 1. If the tree is empty, return a new,

    # single node

**if** node **is** None:

**return** (Node(data))

**else**:

        # 2. Otherwise, recur down the tree

**if** data <**=** node.data:

            node.left **=** insert(node.left, data)

**else**:

            node.right **=** insert(node.right, data)

        # Return the (unchanged) node pointer

**return** node

""" Given a non-empty binary search tree,

return the minimum data value found in that

tree. Note that the entire tree does not need

to be searched. """

**def** minValue(node):

    current **=** node

    # loop down to find the leftmost leaf

**while**(current.left **is not** None):

        current **=** current.left

**return** current.data

# Driver code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

  root **=** None

  root **=** insert(root, 4)

  insert(root, 2)

  insert(root, 1)

  insert(root, 3)

  insert(root, 6)

  insert(root, 5)

  # Function call

**print**("\nMinimum value in BST is %d" **%** (minValue(root)))

# This code is contributed by Nikhil Kumar Singh(nickzuck\_007)

**Output**

Minimum value in BST is 1

**Time Complexity:**O(Height of the BST)

**Auxiliary Space:** O(1)

*From <*[*https://www.geeksforgeeks.org/find-the-minimum-element-in-a-binary-search-tree/*](https://www.geeksforgeeks.org/find-the-minimum-element-in-a-binary-search-tree/)*>*

**Check if an array represents Inorder of Binary Search tree or not**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 01 Dec, 2022
* Read
* Discuss(2)
* Courses
* Practice
* Video

Given an array of **N** element. The task is to check if it is Inorder traversal of any Binary Search Tree or not. Print “Yes” if it is Inorder traversal of any Binary Search Tree else print “No”.

**Examples:**

Input : arr[] = { 19, 23, 25, 30, 45 }  
Output : Yes

Input : arr[] = { 19, 23, 30, 25, 45 }  
Output : No

Recommended Problem

Inorder Traversal and BST

[Binary Search Tree](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Binary%20Search%20Tree&sortBy=submissions)

[Tree](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Tree&sortBy=submissions)

+1 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/inorder-traversal-and-bst5855/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 19.9K

The idea is to use the fact that the inorder traversal of Binary Search Tree is sorted. So, just check if given array is sorted or not.

**Implementation:**

* C++
* Java
* Python3
* C#
* PHP
* Javascript

# Python 3 program to check if a given array

# is sorted or not.

# Function that returns true if array is Inorder

# traversal of any Binary Search Tree or not.

**def** isInorder(arr, n):

    # Array has one or no element

**if** (n **==** 0 **or** n **==** 1):

**return** True

**for** i **in** range(1, n, 1):

        # Unsorted pair found

**if** (arr[i **-** 1] > arr[i]):

**return** False

    # No unsorted pair found

**return** True

# Driver code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    arr **=** [19, 23, 25, 30, 45]

    n **=** len(arr)

**if** (isInorder(arr, n)):

        print("Yes")

**else**:

        print("No")

# This code is contributed by

# Sahil\_Shelangia

**Output**

Yes

**Time complexity: O(n)** where n is the size of array

**Auxiliary Space: O(1)**

*From <*[*https://www.geeksforgeeks.org/check-array-represents-inorder-binary-search-tree-not/*](https://www.geeksforgeeks.org/check-array-represents-inorder-binary-search-tree-not/)*>*

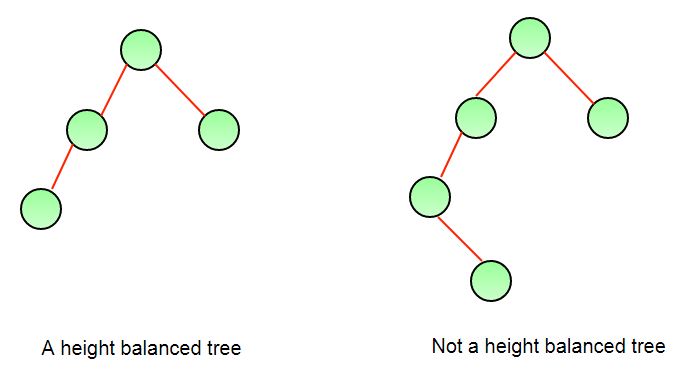
**How to determine if a binary tree is height-balanced?**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 14 Oct, 2022
* Read
* Discuss(305)
* Courses
* Practice
* Video

*A*[***height balanced binary tree***](https://www.geeksforgeeks.org/introduction-to-height-balanced-binary-tree/)*is a binary tree in which the height of the left subtree and right subtree of any node does not differ by more than 1 and both the left and right subtree are also height balanced.*

In this article, we will look into methods how to determine if given Binary trees are height-balanced

***Examples:****The tree on the left is a height balanced binary tree. Whereas the tree on the right is not a height balanced tree. Because the left subtree of the root has a height which is 2 more than the height of the right subtree.*



Recommended Problem

Check for Balanced Tree

[Tree](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Tree&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon&sortBy=submissions)

[Walmart](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Walmart&sortBy=submissions)

+1 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/check-for-balanced-tree/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 1.9L

**Naive Approach:** To check if a tree is height-balanced:

*Get the height of left and right subtrees using*[***dfs***](http://www.geeksforgeeks.org/depth-first-traversal-for-a-graph/)*traversal. Return true if the difference between heights is not more than 1 and left and right subtrees are balanced, otherwise return false.*

Below is the implementation of the above approach.

* C++
* C
* Java
* Python3
* C#
* Javascript

"""

Python3 program to check if a tree is height-balanced

"""

# A binary tree Node

**class** Node:

    # Constructor to create a new Node

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.left **=** None

        self.right **=** None

# function to find height of binary tree

**def** height(root):

    # base condition when binary tree is empty

**if** root **is** None:

**return** 0

**return** max(height(root.left), height(root.right)) **+** 1

# function to check if tree is height-balanced or not

**def** isBalanced(root):

    # Base condition

**if** root **is** None:

**return** True

    # for left and right subtree height

    lh **=** height(root.left)

    rh **=** height(root.right)

    # allowed values for (lh - rh) are 1, -1, 0

**if** (abs(lh **-** rh) <**=** 1) **and** isBalanced(

            root.left) **is** True **and** isBalanced(root.right) **is** True:

**return** True

    # if we reach here means tree is not

    # height-balanced tree

**return** False

# Driver function to test the above function

root **=** Node(1)

root.left **=** Node(2)

root.right **=** Node(3)

root.left.left **=** Node(4)

root.left.right **=** Node(5)

root.left.left.left **=** Node(8)

**if** isBalanced(root):

**print**("Tree is balanced")

**else**:

    print("Tree is not balanced")

# This code is contributed by Shweta Singh

**Output**

Tree is not balanced

**Time Complexity:** O(n^2) in case of [full binary tree](https://www.geeksforgeeks.org/binary-tree-set-3-types-of-binary-tree/).

**Auxiliary Space**: O(n) space for call stack since using recursion

**Efficient implementation:** Above implementation can be optimized by

*Calculating the height in the same recursion rather than calling a height() function separately.*

* For each node make two recursion calls – one for left subtree and the other for the right subtree.
* Based on the heights returned from the recursion calls, decide if the subtree whose root is the current node is height-balanced or not.
* If it is balanced then return the height of that subtree. Otherwise, return **-1** to denote that the subtree is not height-balanced.

Below is the implementation of the above approach.

* C++
* C
* Java
* Python3
* C#
* Javascript

"""

Python3 program to check if a tree is height-balanced

"""

# A binary tree Node

**class** Node:

    # Constructor to create a new Node

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.left **=** None

        self.right **=** None

# Function to check if tree is height-balanced or not

**def** isBalanced(root):

    # Base condition

**if** root **is** None:

**return** True

    # Compute height of left subtree

    lh **=** isBalanced(root.left)

    # If left subtree is not balanced,

    # return -1

**if** lh **== -**1:

**return -**1

    # Do same thing for the right subtree

    rh **=** isBalanced(root.right)

**if** rh **== -**1:

**return -**1

    # Allowed values for (lh - rh) are 1, -1, 0

**if** (abs(lh **-** rh) > 1):

**return -**1

    # If we reach here means tree is

    # height-balanced tree, return height

    # in this case

**else**:

**return** max(lh, rh) **+** 1

# Driver code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    root **=** Node(10)

    root.left **=** Node(5)

    root.right **=** Node(30)

    root.right.left **=** Node(15)

    root.right.right **=** Node(20)

**if** (isBalanced(root) **== -**1):

**print**("Not Balanced")

**else**:

**print**("Balanced")

# This code is contributed by Shweta Singh

**Output**

Balanced

**Time Complexity:** O(n)

* Because we are only one dfs call and utilizing the height returned from that to determine the height balance, it is performing the task in linear time.

*From <*[*https://www.geeksforgeeks.org/how-to-determine-if-a-binary-tree-is-balanced/*](https://www.geeksforgeeks.org/how-to-determine-if-a-binary-tree-is-balanced/)*>*

**Sorted Array to Balanced BST**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 15 Dec, 2022
* Read
* Discuss(130)
* Courses
* Practice
* Video

Given a sorted array. Write a function that creates a Balanced Binary Search Tree using array elements.

**Examples:**

***Input:****arr[] = {1, 2, 3}*

***Output:****A Balanced BST*

*2*

*/  \*

*1     3*

***Explanation:****all elements less than 2 are on the left side of****2****, and all the elements greater than****2****are on the right side*

***Input:****arr[] = {1, 2, 3, 4}*

***Output:****A Balanced BST*

*3*

*/  \*

*2    4*

*/*

*1*

Recommended Problem

Array to BST

[Binary Search Tree](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Binary%20Search%20Tree&sortBy=submissions)

[Tree](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Tree&sortBy=submissions)

+1 more

[Adobe](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Adobe&sortBy=submissions)

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon&sortBy=submissions)

+4 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/array-to-bst4443/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 49.3K

**Sorted Array to Balanced BST By Finding The middle element**

*The idea is to find the middle element of the array and make it the root of the tree, then perform the same operation on the left subarray for the root’s left child and the same operation on the right subarray for the root’s right child.*

Follow the steps mentioned below to implement the approach:

* Set The middle element of the array as root.
* Recursively do the same for the left half and right half.
* Get the middle of the left half and make it the left child of the root created in step 1.
* Get the middle of the right half and make it the right child of the root created in step 1.
* Print the preorder of the tree.

Below is the implementation of the above approach:

* C++
* C
* Java
* Python
* C#
* Javascript

# Python code to convert a sorted array

# to a balanced Binary Search Tree

# binary tree node

**class** Node:

**def** \_\_init\_\_(self, d):

        self.data **=** d

        self.left **=** None

        self.right **=** None

# function to convert sorted array to a

# balanced BST

# input : sorted array of integers

# output: root node of balanced BST

**def** sortedArrayToBST(arr):

**if not** arr:

**return** None

    # find middle index

    mid **=** (len(arr)) **//** 2

    # make the middle element the root

    root **=** Node(arr[mid])

    # left subtree of root has all

    # values <arr[mid]

    root.left **=** sortedArrayToBST(arr[:mid])

    # right subtree of root has all

    # values >arr[mid]

    root.right **=** sortedArrayToBST(arr[mid**+**1:])

**return** root

# A utility function to print the preorder

# traversal of the BST

**def** preOrder(node):

**if not** node:

**return**

**print** node.data,

    preOrder(node.left)

    preOrder(node.right)

# driver program to test above function

"""

Constructed balanced BST is

    4

/ \

2 6

/ \ / \

1 3 5 7

"""

arr **=** [1, 2, 3, 4, 5, 6, 7]

root **=** sortedArrayToBST(arr)

**print** "PreOrder Traversal of constructed BST ",

preOrder(root)

# This code is contributed by Ishita Tripathi

**Output**

PreOrder Traversal of constructed BST   
4 2 1 3 6 5 7

**Time Complexity:**O(N)

**Auxiliary Space:**O(H) ~= O(log(N)), for recursive stack space where H is the height of the tree.

*From <*[*https://www.geeksforgeeks.org/sorted-array-to-balanced-bst/*](https://www.geeksforgeeks.org/sorted-array-to-balanced-bst/)*>*

**Check for Identical BSTs without building the trees**

* Difficulty Level : [Hard](https://www.geeksforgeeks.org/hard/)
* Last Updated : 25 Oct, 2022
* Read
* Discuss(217)
* Courses
* Practice
* Video

Given two arrays that represent a sequence of keys. Imagine we make a Binary Search Tree (BST) from each array. We need to tell whether two BSTs will be identical or not without actually constructing the tree.

**Example:**

For example, the input arrays are {2, 4, 3, 1} and {2, 1, 4, 3} will construct the same tree

Let the input arrays be a[] and b[]

Example 1:  
a[] = {2, 4, 1, 3} will construct following tree.  
 2  
 / \  
1 4  
 /  
 3  
b[] = {2, 4, 3, 1} will also construct the same tree.  
 2  
 / \  
1 4  
 /  
 3   
So the output is "True"

Example 2:  
a[] = {8, 3, 6, 1, 4, 7, 10, 14, 13}  
b[] = {8, 10, 14, 3, 6, 4, 1, 7, 13}

They both construct the same following BST, so output is "True"  
 8  
 / \  
 3 10  
 / \ \  
 1 6 14  
 / \ /  
 4 7 13

**Solution:**

* According to BST property, elements of the left subtree must be smaller and elements of right subtree must be greater than root.
* Two arrays represent the same BST if, for every element x, the elements in left and right subtrees of x appear after it in both arrays. And same is true for roots of left and right subtrees.
* The idea is to check of if next smaller and greater elements are same in both arrays. Same properties are recursively checked for left and right subtrees. The idea looks simple, but implementation requires checking all conditions for all elements. Following is an interesting recursive implementation of the idea.

**Implementation:**

* C++
* C
* Java
* Python3
* C#
* Javascript

# A Python3 program to check for Identical

# BSTs without building the trees

# # The main function that checks if two

# arrays a[] and b[] of size n construct

# same BST. The two values 'min' and 'max'

# decide whether the call is made for left

# subtree or right subtree of a parent

# element. The indexes i1 and i2 are the

# indexes in (a[] and b[]) after which we

# search the left or right child. Initially,

# the call is made for INT\_MIN and INT\_MAX

# as 'min' and 'max' respectively, because

# root has no parent. i1 and i2 are just

# after the indexes of the parent element in a[] and b[]. \*/

**def** isSameBSTUtil(a, b, n, i1, i2, min, max):

    # # Search for a value satisfying the

    # constraints of min and max in a[] and

    # b[]. If the parent element is a leaf

    # node then there must be some elements

    # in a[] and b[] satisfying constraint. \*/

    j, k **=** i1, i2

**while** j < n:

**if** (a[j] > min **and** a[j] < max):

**break**;

        j **+=** 1

**while** k<n:

**if** (b[k] > min **and** b[k] < max):

**break**

        k **+=** 1

    # If the parent element is leaf in both arrays \*/

**if** (j **==** n **and** k **==** n):

**return** True

    # Return false if any of the following is true

        # a) If the parent element is leaf in one array,

        #     but non-leaf in other.

        # b) The elements satisfying constraints are

        #     not same. We either search for left

        #     child or right child of the parent

        #     element (decided by min and max values).

        #     The child found must be same in both arrays \*/

**if** (((j **==** n) ^ (k **==** n)) **or** a[j] !**=** b[k]):

**return** False

    # Make the current child as parent and

    # recursively check for left and right

    # subtrees of it. Note that we can also

    # pass a[k] in place of a[j] as they

    # are both are same \*/

**return** isSameBSTUtil(a, b, n, j **+** 1, k **+** 1, a[j], max) **and** isSameBSTUtil(a, b, n, j **+** 1, k **+** 1, min, a[j]) #Left Subtree

# A wrapper over isSameBSTUtil()

**def** isSameBST(a, b, n):

**return** isSameBSTUtil(a, b, n, 0, 0, **-**10**\*\***9, 10**\*\***9)

# Driver code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    a **=** [8, 3, 6, 1, 4, 7, 10, 14, 13]

    b **=** [8, 10, 14, 3, 6, 4, 1, 7, 13]

    n **=** len(a)

**if**(isSameBST(a, b, n)):

        print("BSTs are same")

**else**:

        print("BSTs not same")

# This code is contributed by mohit kumar 29.

**Output**

BSTs are same

**Time Complexity:**O(N2)

**Auxiliary Space:**O(N), for recursive stack space.

*From <*[*https://www.geeksforgeeks.org/check-for-identical-bsts-without-building-the-trees/*](https://www.geeksforgeeks.org/check-for-identical-bsts-without-building-the-trees/)*>*

**Check for Identical BSTs without building the trees**

* Difficulty Level : [Hard](https://www.geeksforgeeks.org/hard/)
* Last Updated : 25 Oct, 2022
* Read
* Discuss(217)
* Courses
* Practice
* Video

Given two arrays that represent a sequence of keys. Imagine we make a Binary Search Tree (BST) from each array. We need to tell whether two BSTs will be identical or not without actually constructing the tree.

**Example:**

For example, the input arrays are {2, 4, 3, 1} and {2, 1, 4, 3} will construct the same tree

Let the input arrays be a[] and b[]

Example 1:  
a[] = {2, 4, 1, 3} will construct following tree.  
 2  
 / \  
1 4  
 /  
 3  
b[] = {2, 4, 3, 1} will also construct the same tree.  
 2  
 / \  
1 4  
 /  
 3   
So the output is "True"

Example 2:  
a[] = {8, 3, 6, 1, 4, 7, 10, 14, 13}  
b[] = {8, 10, 14, 3, 6, 4, 1, 7, 13}

They both construct the same following BST, so output is "True"  
 8  
 / \  
 3 10  
 / \ \  
 1 6 14  
 / \ /  
 4 7 13

**Solution:**

* According to BST property, elements of the left subtree must be smaller and elements of right subtree must be greater than root.
* Two arrays represent the same BST if, for every element x, the elements in left and right subtrees of x appear after it in both arrays. And same is true for roots of left and right subtrees.
* The idea is to check of if next smaller and greater elements are same in both arrays. Same properties are recursively checked for left and right subtrees. The idea looks simple, but implementation requires checking all conditions for all elements. Following is an interesting recursive implementation of the idea.

**Implementation:**

* C++
* C
* Java
* Python3
* C#
* Javascript

# A Python3 program to check for Identical

# BSTs without building the trees

# # The main function that checks if two

# arrays a[] and b[] of size n construct

# same BST. The two values 'min' and 'max'

# decide whether the call is made for left

# subtree or right subtree of a parent

# element. The indexes i1 and i2 are the

# indexes in (a[] and b[]) after which we

# search the left or right child. Initially,

# the call is made for INT\_MIN and INT\_MAX

# as 'min' and 'max' respectively, because

# root has no parent. i1 and i2 are just

# after the indexes of the parent element in a[] and b[]. \*/

**def** isSameBSTUtil(a, b, n, i1, i2, min, max):

    # # Search for a value satisfying the

    # constraints of min and max in a[] and

    # b[]. If the parent element is a leaf

    # node then there must be some elements

    # in a[] and b[] satisfying constraint. \*/

    j, k **=** i1, i2

**while** j < n:

**if** (a[j] > min **and** a[j] < max):

**break**;

        j **+=** 1

**while** k<n:

**if** (b[k] > min **and** b[k] < max):

**break**

        k **+=** 1

    # If the parent element is leaf in both arrays \*/

**if** (j **==** n **and** k **==** n):

**return** True

    # Return false if any of the following is true

        # a) If the parent element is leaf in one array,

        #     but non-leaf in other.

        # b) The elements satisfying constraints are

        #     not same. We either search for left

        #     child or right child of the parent

        #     element (decided by min and max values).

        #     The child found must be same in both arrays \*/

**if** (((j **==** n) ^ (k **==** n)) **or** a[j] !**=** b[k]):

**return** False

    # Make the current child as parent and

    # recursively check for left and right

    # subtrees of it. Note that we can also

    # pass a[k] in place of a[j] as they

    # are both are same \*/

**return** isSameBSTUtil(a, b, n, j **+** 1, k **+** 1, a[j], max) **and** isSameBSTUtil(a, b, n, j **+** 1, k **+** 1, min, a[j]) #Left Subtree

# A wrapper over isSameBSTUtil()

**def** isSameBST(a, b, n):

**return** isSameBSTUtil(a, b, n, 0, 0, **-**10**\*\***9, 10**\*\***9)

# Driver code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    a **=** [8, 3, 6, 1, 4, 7, 10, 14, 13]

    b **=** [8, 10, 14, 3, 6, 4, 1, 7, 13]

    n **=** len(a)

**if**(isSameBST(a, b, n)):

        print("BSTs are same")

**else**:

        print("BSTs not same")

# This code is contributed by mohit kumar 29.

**Output**

BSTs are same

**Time Complexity:**O(N2)

**Auxiliary Space:**O(N), for recursive stack space.

*From <*[*https://www.geeksforgeeks.org/check-for-identical-bsts-without-building-the-trees/*](https://www.geeksforgeeks.org/check-for-identical-bsts-without-building-the-trees/)*>*

**Convert BST to Min Heap**

Given a binary search tree which is also a [complete binary tree](http://www.geeksforgeeks.org/binary-tree-set-3-types-of-binary-tree/). The problem is to **convert the given BST into a Min Heap** with the condition that **all the values in the left subtree of a node should be less than all the values in the right subtree of the node**. This condition is applied to all the nodes, in the resultant converted Min Heap.

**Examples:**

***Input:****4*

*/   \*

*2     6*

*/  \   /  \*

*1   3  5    7*

***Output:****1*

*/   \*

*2     5*

*/  \   /  \*

*3   4  6    7*

***Explanation:****The given****BST****has been transformed into a****Min Heap.****All the nodes in the Min Heap satisfies the given condition, that is, values in the left subtree of a node should be less than the values in the right subtree of the node.*

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Approach:**To solve the problem using this approach follow the below idea:

*Store the inorder traversal of the BST in array and then do preorder traversal of the BST and while doing preorder traversal copy the values of inorder traversal into the current node, as copying the sorted elements while doing preorder traversal will make sure that a Min-Heap is constructed with the condition that all the values in the left subtree of a node are less than all the values in the right subtree of the node.*

Follow the given steps to solve the problem:

* Create an array arr[] of size **N**, where N is the number of nodes in the given BST.
* Perform the inorder traversal of the BST and copy the node values in the arr[] in sorted order.
* Now perform the preorder traversal of the tree.
* While traversing the root during the preorder traversal, one by one copy the values from the array arr[] to the nodes of the BST.

Below is the implementation of the above approach:

# Python3 implementation to convert the

# given BST to Min Heap

# structure of a node of BST

**class** Node:

    # Constructor to create a new node

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.left **=** None

        self.right **=** None

# function for the inorder traversal

# of the tree so as to store the node

# values in 'arr' in sorted order

**def** inorderTraversal(root, arr):

**if** root **==** None:

**return**

    # first recur on left subtree

    inorderTraversal(root.left, arr)

    # then copy the data of the node

    arr.append(root.data)

    # now recur for right subtree

    inorderTraversal(root.right, arr)

# function to convert the given

# BST to MIN HEAP performs preorder

# traversal of the tree

**def** BSTToMinHeap(root, arr, i):

**if** root **==** None:

**return**

    # first copy data at index 'i' of

    # 'arr' to the node

    i[0] **+=** 1

    root.data **=** arr[i[0]]

    # then recur on left subtree

    BSTToMinHeap(root.left, arr, i)

    # now recur on right subtree

    BSTToMinHeap(root.right, arr, i)

# utility function to convert the

# given BST to MIN HEAP

**def** convertToMinHeapUtil(root):

    # vector to store the data of

    # all the nodes of the BST

    arr **=** []

    i **=** [**-**1]

    # inorder traversal to populate 'arr'

    inorderTraversal(root, arr)

    # BST to MIN HEAP conversion

    BSTToMinHeap(root, arr, i)

# function for the preorder traversal

# of the tree

**def** preorderTraversal(root):

**if** root **==** None:

**return**

    # first print the root's data

    print(root.data, end**=**" ")

    # then recur on left subtree

    preorderTraversal(root.left)

    # now recur on right subtree

    preorderTraversal(root.right)

# Driver's Code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    # BST formation

    root **=** Node(4)

    root.left **=** Node(2)

    root.right **=** Node(6)

    root.left.left **=** Node(1)

    root.left.right **=** Node(3)

    root.right.left **=** Node(5)

    root.right.right **=** Node(7)

    # Function call

    convertToMinHeapUtil(root)

**print**("Preorder Traversal:")

    preorderTraversal(root)

# This code is contributed

# by PranchalK

**Output**

Preorder Traversal:  
1 2 3 4 5 6 7

**Time Complexity:** O(N)

**Auxiliary Space:** O(N)

*From <*[*https://www.geeksforgeeks.org/convert-bst-min-heap/*](https://www.geeksforgeeks.org/convert-bst-min-heap/)*>*

**Second largest element in BST**

Given a Binary Search Tree(BST), find the second largest element.

**Examples:**

Input: Root of below BST  
 10  
 /  
 5

Output: 5

Input: Root of below BST  
 10  
 / \  
 5 20  
 \   
 30

Output: 20

Source: [Microsoft Interview](https://www.geeksforgeeks.org/microsoft-interview-experience-set-53/)

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

The idea is similar to below post.

[K’th Largest Element in BST when modification to BST is not allowed](https://www.geeksforgeeks.org/kth-largest-element-in-bst-when-modification-to-bst-is-not-allowed/)

The second largest element is second last element in inorder traversal and second element in reverse inorder traversal. We traverse given Binary Search Tree in reverse inorder and keep track of counts of nodes visited. Once the count becomes 2, we print the node.

Below is the implementation of above idea.

* C++
* Java
* Python3
* C#
* Javascript

# Python3 code to find second largest

# element in BST

**class** Node:

    # Constructor to create a new node

**def** \_\_init\_\_(self, data):

        self.key **=** data

        self.left **=** None

        self.right **=** None

# A function to find 2nd largest

# element in a given tree.

**def** secondLargestUtil(root, c):

    # Base cases, the second condition

    # is important to avoid unnecessary

    # recursive calls

**if** root **==** None **or** c[0] >**=** 2:

**return**

    # Follow reverse inorder traversal so that

    # the largest element is visited first

    secondLargestUtil(root.right, c)

    # Increment count of visited nodes

    c[0] **+=** 1

    # If c becomes k now, then this is

    # the 2nd largest

**if** c[0] **==** 2:

**print**("2nd largest element is",

                              root.key)

**return**

    # Recur for left subtree

    secondLargestUtil(root.left, c)

# Function to find 2nd largest element

**def** secondLargest(root):

    # Initialize count of nodes

    # visited as 0

    c **=** [0]

    # Note that c is passed by reference

    secondLargestUtil(root, c)

# A utility function to insert a new

# node with given key in BST

**def** insert(node, key):

    # If the tree is empty, return a new node

**if** node **==** None:

**return** Node(key)

    # Otherwise, recur down the tree

**if** key < node.key:

        node.left **=** insert(node.left, key)

**elif** key > node.key:

        node.right **=** insert(node.right, key)

    # return the (unchanged) node pointer

**return** node

# Driver Code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    # Let us create following BST

    #         50

    #     /     \

    #     30     70

    #     / \ / \

    # 20 40 60 80

    root **=** None

    root **=** insert(root, 50)

    insert(root, 30)

    insert(root, 20)

    insert(root, 40)

    insert(root, 70)

    insert(root, 60)

    insert(root, 80)

    secondLargest(root)

# This code is contributed by PranchalK

**Output:**

2nd largest element is 70

**Time complexity** : O(h) where h is height of BST.

**Space Complexity**: O(n) for call stack where n is total no of nodes in BST

*From <*[*https://www.geeksforgeeks.org/second-largest-element-in-binary-search-tree-bst/*](https://www.geeksforgeeks.org/second-largest-element-in-binary-search-tree-bst/)*>*

So to traverse the BST in descending order we use reverse in-order traversal of BST. This takes a global variable sum which is updated at every node and once the root node is reached it is added to the value of root node and value of the root node is updated.

* C++
* C
* Java
* Python3
* C#
* Javascript

# Python3 program to add all greater values

# in every node of BST

# A utility function to create a

# new BST node

**class** newNode:

    # Constructor to create a new node

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.left **=** None

        self.right **=** None

# Recursive function to add all greater

# values in every node

**def** modifyBSTUtil(root, Sum):

    # Base Case

**if** root **==** None:

**return**

    # Recur for right subtree

    modifyBSTUtil(root.right, Sum)

    # Now Sum[0] has sum of nodes in right

    # subtree, add root.data to sum and

    # update root.data

    Sum[0] **=** Sum[0] **+** root.data

    root.data **=** Sum[0]

    # Recur for left subtree

    modifyBSTUtil(root.left, Sum)

# A wrapper over modifyBSTUtil()

**def** modifyBST(root):

    Sum **=** [0]

    modifyBSTUtil(root, Sum)

# A utility function to do inorder

# traversal of BST

**def** inorder(root):

**if** root !**=** None:

        inorder(root.left)

        print(root.data, end **=**" ")

        inorder(root.right)

# A utility function to insert a new node

# with given data in BST

**def** insert(node, data):

    # If the tree is empty, return a new node

**if** node **==** None:

**return** newNode(data)

    # Otherwise, recur down the tree

**if** data <**=** node.data:

        node.left **=** insert(node.left, data)

**else**:

        node.right **=** insert(node.right, data)

    # return the (unchanged) node pointer

**return** node

# Driver Code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    # Let us create following BST

    # 50

    #     /     \

    # 30     70

    #     / \ / \

    # 20 40 60 80

    root **=** None

    root **=** insert(root, 50)

    insert(root, 30)

    insert(root, 20)

    insert(root, 40)

    insert(root, 70)

    insert(root, 60)

    insert(root, 80)

    modifyBST(root)

    # print inorder traversal of the

    # modified BST

    inorder(root)

# This code is contributed by PranchalK

**Output**

350 330 300 260 210 150 80

**Complexity Analysis:**

* **Time Complexity:** O(n).   
  As this problem uses an in-order tree traversal technique
* **Auxiliary Space:** O(1).   
  As no data structure has been used for storing values.

*From <*[*https://www.geeksforgeeks.org/add-greater-values-every-node-given-bst/*](https://www.geeksforgeeks.org/add-greater-values-every-node-given-bst/)*>*

 temp.val **=** val1

    temp.left **=** temp.right **=** None

**return** temp

s **=** {}

# function to insert elements of the

# tree to map m

**def** insertToHash(root):

**if** (root **==** None):

**return**

    insertToHash(root.left)

    s.add(root.data)

    insertToHash(root.right)

# function to check if the two BSTs contain

# same set of elements

**def** checkBSTs(root1, root2):

    # Base cases

**if** (root1 !**=** None **and** root2 !**=** None) :

**return** True

**if** ((root1 **==** None **and** root2 !**=** None) **or**

        (root1 !**=** None **and** root2 **==** None)):

**return** False

    # Create two hash sets and store

    # elements both BSTs in them.

    s1 **=** {}

    s2 **=** {}

    s **=** s1

    insertToHash(root1)

    s1 **=** s

    s **=** s2

    insertToHash(root2)

    s2 **=** s

    # Return True if both hash sets

    # contain same elements.

**return** (s1 **==** (s2))

# Driver code

# First BST

root1 **=** Node\_(15)

root1.left **=** Node\_(10)

root1.right **=** Node\_(20)

root1.left.left **=** Node\_(5)

root1.left.right **=** Node\_(12)

root1.right.right **=** Node\_(25)

# Second BST

root2 **=** Node\_(15)

root2.left **=** Node\_(12)

root2.right **=** Node\_(20)

root2.left.left **=** Node\_(5)

root2.left.left.right **=** Node\_(10)

root2.right.right **=** Node\_(25)

# check if two BSTs have same set of elements

**if** (checkBSTs(root1, root2)):

    print("YES")

**else**:

    print("NO")

# This code is contributed by Arnab Kundu

**Output**

YES

**Time Complexity: O( n ),** where n is the number of nodes in the trees.

**Auxiliary Space: O( n )**.

**Method 3**: We know about an interesting property of BST that inorder traversal of a BST generates a sorted array. So we can do inorder traversals of both the BSTs and generate two arrays and finally, we can compare these two arrays. If both of the arrays are the same then the BSTs have the same set of elements otherwise not.

**Implementation:**

# Python3 program to check if two BSTs contains

# same set of elements

# BST Node

**class** Node:

**def** \_\_init\_\_(self):

        self.data **=** 0

        self.left **=** None

        self.right **=** None

# Utility function to create Node

**def** Node\_(val1):

    temp **=** Node()

    temp.data **=** val1

    temp.left **=** temp.right **=** None

**return** temp

v **=** []

# function to insert elements of the

# tree to map m

**def** storeInorder(root):

**if** (root **==** None):

**return**

    storeInorder(root.left)

    v.append(root.data)

    storeInorder(root.right)

# function to check if the two BSTs contain

# same set of elements

**def** checkBSTs(root1, root2):

    # Base cases

**if** (root1 **==** None **and** root2 **==** None) :

**return** True

**if** ((root1 **==** None **and** root2 !**=** None) **or** \

        (root1 !**=** None **and** root2 **==** None)):

**return** False

    # Create two hash sets and store

    # elements both BSTs in them.

    v1 **=** []

    v2 **=** []

    v **=** v1

    storeInorder(root1)

    v1 **=** v

    v **=** v2

    storeInorder(root2)

    v2 **=** v

    # Return True if both hash sets

    # contain same elements.

**return** (v1 **==** v2)

# Driver code

# First BST

root1 **=** Node\_(15)

root1.left **=** Node\_(10)

root1.right **=** Node\_(20)

root1.left.left **=** Node\_(5)

root1.left.right **=** Node\_(12)

root1.right.right **=** Node\_(25)

# Second BST

root2 **=** Node\_(15)

root2.left **=** Node\_(12)

root2.right **=** Node\_(20)

root2.left.left **=** Node\_(5)

root2.left.left.right **=** Node\_(10)

root2.right.right **=** Node\_(25)

# check if two BSTs have same set of elements

**if** (checkBSTs(root1, root2)):

    print("YES")

**else**:

    print("NO")

# This code is contributed by SHUBHAMSINGH10

**Output**

YES

**Time Complexity: O( n ).**

**Auxiliary Space: O( n ).**

*From <*[*https://www.geeksforgeeks.org/check-two-bsts-contain-set-elements/*](https://www.geeksforgeeks.org/check-two-bsts-contain-set-elements/)*>*

 temp.val **=** val1

    temp.left **=** temp.right **=** None

**return** temp

s **=** {}

# function to insert elements of the

# tree to map m

**def** insertToHash(root):

**if** (root **==** None):

**return**

    insertToHash(root.left)

    s.add(root.data)

    insertToHash(root.right)

# function to check if the two BSTs contain

# same set of elements

**def** checkBSTs(root1, root2):

    # Base cases

**if** (root1 !**=** None **and** root2 !**=** None) :

**return** True

**if** ((root1 **==** None **and** root2 !**=** None) **or**

        (root1 !**=** None **and** root2 **==** None)):

**return** False

    # Create two hash sets and store

    # elements both BSTs in them.

    s1 **=** {}

    s2 **=** {}

    s **=** s1

    insertToHash(root1)

    s1 **=** s

    s **=** s2

    insertToHash(root2)

    s2 **=** s

    # Return True if both hash sets

    # contain same elements.

**return** (s1 **==** (s2))

# Driver code

# First BST

root1 **=** Node\_(15)

root1.left **=** Node\_(10)

root1.right **=** Node\_(20)

root1.left.left **=** Node\_(5)

root1.left.right **=** Node\_(12)

root1.right.right **=** Node\_(25)

# Second BST

root2 **=** Node\_(15)

root2.left **=** Node\_(12)

root2.right **=** Node\_(20)

root2.left.left **=** Node\_(5)

root2.left.left.right **=** Node\_(10)

root2.right.right **=** Node\_(25)

# check if two BSTs have same set of elements

**if** (checkBSTs(root1, root2)):

    print("YES")

**else**:

    print("NO")

# This code is contributed by Arnab Kundu

**Output**

YES

**Time Complexity: O( n ),** where n is the number of nodes in the trees.

**Auxiliary Space: O( n )**.

**Method 3**: We know about an interesting property of BST that inorder traversal of a BST generates a sorted array. So we can do inorder traversals of both the BSTs and generate two arrays and finally, we can compare these two arrays. If both of the arrays are the same then the BSTs have the same set of elements otherwise not.

**Implementation:**

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to check if two BSTs contains

# same set of elements

# BST Node

**class** Node:

**def** \_\_init\_\_(self):

        self.data **=** 0

        self.left **=** None

        self.right **=** None

# Utility function to create Node

**def** Node\_(val1):

    temp **=** Node()

    temp.data **=** val1

    temp.left **=** temp.right **=** None

**return** temp

v **=** []

# function to insert elements of the

# tree to map m

**def** storeInorder(root):

**if** (root **==** None):

**return**

    storeInorder(root.left)

    v.append(root.data)

    storeInorder(root.right)

# function to check if the two BSTs contain

# same set of elements

**def** checkBSTs(root1, root2):

    # Base cases

**if** (root1 **==** None **and** root2 **==** None) :

**return** True

**if** ((root1 **==** None **and** root2 !**=** None) **or** \

        (root1 !**=** None **and** root2 **==** None)):

**return** False

    # Create two hash sets and store

    # elements both BSTs in them.

    v1 **=** []

    v2 **=** []

    v **=** v1

    storeInorder(root1)

    v1 **=** v

    v **=** v2

    storeInorder(root2)

    v2 **=** v

    # Return True if both hash sets

    # contain same elements.

**return** (v1 **==** v2)

# Driver code

# First BST

root1 **=** Node\_(15)

root1.left **=** Node\_(10)

root1.right **=** Node\_(20)

root1.left.left **=** Node\_(5)

root1.left.right **=** Node\_(12)

root1.right.right **=** Node\_(25)

# Second BST

root2 **=** Node\_(15)

root2.left **=** Node\_(12)

root2.right **=** Node\_(20)

root2.left.left **=** Node\_(5)

root2.left.left.right **=** Node\_(10)

root2.right.right **=** Node\_(25)

# check if two BSTs have same set of elements

**if** (checkBSTs(root1, root2)):

    print("YES")

**else**:

    print("NO")

# This code is contributed by SHUBHAMSINGH10

**Output**

YES

**Time Complexity: O( n ).**

**Auxiliary Space: O( n ).**