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

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

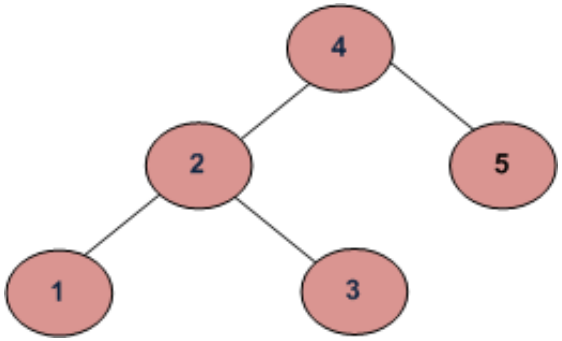
• The left subtree of a node contains only nodes with keys less than the node's key.

• 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.




**METHOD 1 (Simple but Wrong)**

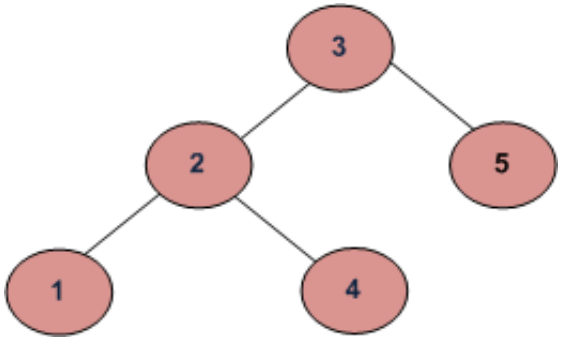
Following is a simple program. For each node, check if left node of it is smaller than the node and right node of it is greater than the node.

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

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

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

  /* false if, recursively, the left or right is not a BST */
  if (!isBST(node->left) || !isBST(node->right))
    return 0;

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

Run on IDE

**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.

```
/* Returns true if a binary tree is a binary search tree */
int isBST(struct node* node)
{
```

```
  if (node == NULL)
    return(true);

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

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

  /* false if, recursively, the left or right is not a BST */
  if (!isBST(node->left) || !isBST(node->right))
    return(false);

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

Run on IDE

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)**



de only once. The trick is to
ne narrowing min and max
[_MAX — they narrow from
there.

```
/* Returns true if the given tree is a binary search tree
 (efficient version). */
int isBST(struct 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(struct node* node, int min, int max)
```

**Implementation:**

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

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

int isBSTUtil(struct node* node, int min, int max);

/* Returns true if the given tree is a binary search tree
 (efficient version). */
int isBST(struct 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(struct 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 recursively,
   tightening the min or max constraint */
  return
    isBSTUtil(node->left, min, node->data-1) &&  // Allow only distinct values
    isBSTUtil(node->right, node->data+1, max);  // Allow only distinct values
}

/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct node* newNode(int data)
```

```
{
  struct node* node = (struct node*)
                      malloc(sizeof(struct node));
  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;
}
```

Run on IDE

Time Complexity: O(n)

Auxiliary Space : O(1) if Function Call Stack size is not considered, otherwise O(n)

**METHOD 4(Using In-Order Traversal)**

Thanks to LJW489 for suggesting this method.

1) Do In-Order Traversal of the given tree and store the result in a temp array.

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 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.

```
bool isBST(struct node* root)
```

```
{
    static struct 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;
}
```

Run on IDE

The use of static variable can also be avoided by using reference to prev node as a parameter (Similar to this post).

**Sources:**

http://en.wikipedia.org/wiki/Binary_search_tree

http://cslibrary.stanford.edu/110/BinaryTrees.html

Please write comments if you find any bug in the above programs/algorithms or other ways to solve the same problem.


226 Comments Category: Trees

## Related Questions:

- Construct all possible BSTs for keys 1 to N
- K'th smallest element in BST using O(1) Extra Space
- Count BST subtrees that lie in given range
- Count BST nodes that lie in a given range
- Data Structure for a single resource reservations
- How to handle duplicates in Binary Search Tree?
- Handshaking Lemma and Interesting Tree Properties
- Advantages of BST over Hash Table

Join the discussion…

**Varun Sagar** • 12 days ago

without using any static variable

```
public Boolean isBST(Tree root)
{

if (root == null)
return true;

// adding two ifs for clarity
if (root.left != null && root.right != null )
{
if (root.val < root.left.val || root.val > root.right.val)
return false;
}

return (isBST (root.left) && isBST (root.right));
}
```

• Reply • Share ›

**Avinash Kr Suhjani** • 17 days ago

This is how I solved this problem:

```
int check( struct node *node)
{
if(node->left==NULL)
{
if(node->right->data > node->data)
return 1;

}
```

```
}
if(node->right==NULL)
{
if(node->left->data < node->data)
return 1;
}

if( (node->left->data < node->data) && (node->right->data > node->data) )
{
return 1;
```

see more

• Reply • Share ›

**Avatar** ➚ Avinash Kr Suhjani • 17 days ago

Why it is passing min with data-1 and data+1 at max??

• Reply • Share ›

**Avatar** • 18 days ago

Why it is passing min with data-1 and data+1 at max??

• Reply • Share ›

**Shashi Jey** • 20 days ago

https://ideone.com/kVLYJo

• Reply • Share ›

**Anonymous Face-palm** • 22 days ago

Why we need to do so many dumb things?
A traverse of BST is a sorted list. Traverse, put that tree as a list - check if the darn list is sorted. You are done.

• Reply • Share ›

**Avatar** ➚ Anonymous Face-palm • 17 days ago

Why it is passing min with data-1 and data+1 at max value??

• Reply • Share ›

**suvro** • a month ago

private boolean isBST(Node node) {

if (node==null)

return true;

else if(node.leftChild==null && node.rightChild==null)

return true;

else if(node.leftChild!=null&&node.leftChild.value<node.value){ if(node.rightchild="=null||node.rightChild.value">node.value)

return true;

else

return false;

}else if(node.rightChild!=null&&node.rightChild.value>node.value){

if(node.leftChild==null||node.leftChild.value>node.value)

see more

• Reply • Share ›

**GLOBAL** • a month ago

if (!isBST(root->left))
return false
i didnot understand in last method can anyone explain it
thanks....

• Reply • Share ›

**Dhruv Gosain** ➚ GLOBAL • a month ago

Here a static node pointer is declared which will keep the track of the inorder predecessor of the current root pointer. When the left subtree has been processed completely then it is checked that whether the left subtree preserves the

When the left subtree has been processed completely then it is checked that whether the left subtree preserves the property of the BST and after that it is checked that the same property i maintained by the root and after that the right subtree is checked for the BST property, similar to an inorder fashion.

```
bool isBST(nodeptr head, nodeptr* prev)

{

// if the head is NULL then by default return true

if(head == NULL)

return true;

// check the left subtree recursively

bool flag = isBST(head -> left, prev);

// return false if the left subtree breaks the property
```

see more

2 ^ | v • Reply • Share ›

**Bewkoof_coder** • a month ago

the above problem using inorder traversal..,
Recursive code.. https://ideone.com/wOafVj. O(n) complexity.
Iterative code..https://ideone.com/pPXYY5

^ | v • Reply • Share ›

**JavaCoder** • a month ago

Method 4 seems too complex to understand !!

1 ^ | v • Reply • Share ›

**coderr** • a month ago

Can someone explain why method 2 traverses some part of the tree many times?

Is it because we are using helper functions like minValue() and maxValue()? Correct me if I'm wrong.

What should be the complexity of method 2?

• Reply • Share ›

**#Noso** coderr • a month ago

Yes, You are right .It just because of helper functions like minValue() and maxValue().

And I think time complexity will be as below

for isBST() method it will be O(h) (excluded Min or Max)

And for Min or Max method it will be O(h) as u may need to go till the deepest node in the tree to find out Min or Max.

# So this is something like for loop inside for loop O(h^2) :)

@Think Loud (Noso)

1 • Reply • Share ›

**see_language** • 2 months ago

Why can't we change the if condition to "root->data <= min || root->data >= max" and not add or subtract 1 to root->data?

• Reply • Share ›

**Lokesh** • 2 months ago

http://code.geeksforgeeks.org/...

1 • Reply • Share ›

**Lokesh** Lokesh • 9 days ago

http://code.geeksforgeeks.org/...

• Reply • Share ›

**sid** • 2 months ago

A simple solution :-

initial arguments to be passed :-

ptr = root of the tree
nMin = INT_MIN
nMax = INT_MAX

bool isBST(NodeTree* ptr, int nMin, int nMax)
//
if ptr is equal to null
then
return true
else
return (ptr->data > nMin)
&& (ptr->data < nMax)
&& isBST(ptr->left, nMin, ptr->data)
&& isBST(ptr->right, ptr->data, nMax)

• Reply • Share ›

**Abhishek Jaiswal** • 2 months ago

I think method-4 won't work because value of prev is passed
to the calling recursive function is not passed.
Each recursive function has NULL value stored in its prev.
May if we could declare prev globally that code should work

• Reply • Share ›

**Abhinav** ➚ Abhishek Jaiswal • a month ago

We don't need to pass prev in every function call since we have declared it as a static variable. So, the pointer will retain its value between subsequent function calls.

• Reply • Share ›

**Nimesh Arora** • 2 months ago

i think adding two statement in method 1 will not only correct it but also solve in O(n) , WITH NO MAX OR MIN.
TELL ME IF I M WRONG ???

int isBST(struct node* node)

```
{

if (node == NULL)

return 1;

/* false if left is > than node */

if (node->left != NULL && node->left->data > node->data)

return 0;

/* false if node->left->right is > than node */

if (node->left->right !=NULL && node->left->right->data > node->data)
```

see more

1 • Reply • Share ›

**JavaCoder** ➚ Nimesh Arora • 2 months ago

Nope !

Make a BST with elements in this order: 8, 4, 10, 2, 6, 5, 7
and replace 7 with 9.

Now, check it with your code. Your code returns 1 for this tree, but it is not a BST as node with value 9 is in the left subtree of root node with value 8.

1 • Reply • Share ›

**Nimesh Arora** ➚ JavaCoder • 2 months ago

yes u r right .... thanx

• Reply • Share ›

**Ankit** • 2 months ago

Method 3 will fail : In following test case:

12
/ \
/ \

11 13

for 11 : max value become : 12-1=11

now it will return false as (node->data > max) ie;(11>12);

so it will return false .

• Reply • Share ›

**JavaCoder** ➚ Ankit • 2 months ago

As you calculated, max becomes 11. Now, we check:

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

Since node->data is 11, and 11 > 11 is false, 0 is NOT returned from here.

• Reply • Share ›

**sujeet singh** • 2 months ago

traverse the tree in inorder and check its maintain the ascending order . below is implementation -

```
bool isBSTreeUtility(struct node*btree,int &min,int &max)
{
if(btree)
{
isBSTreeUtility(btree->left,min,max);
if(btree->data > min && btree->data <max) min="btree-">data;
else
return false;
```

```
isBSTreeUtility(btree->right,min,max);
}

return true;

}

bool isBST(struct node *btree)
{
return isBSTreeUtility(btree, INT_MIN, INT_MAX);
}
```

what i have done is traverse the tree in inorder and check if it maintains ascending ...



**Narendra** • 3 months ago

@GeeksforGeeks Another approach
In this we traverse bottom up.
http://ideone.com/wRcBHk



**Varun G** • 4 months ago

http://rawjava.blogspot.in/201...



**Aryan Parker** • 4 months ago

Why is my function not working for a tree like this-
4
/ \
2 6
/ \ /
1 3 5

```
int checkbst(Node root)
```

```
{
Node ptr=root;
if(!ptr)return 1;
if(!ptr->left&&!ptr->right)return 1;
int left_value=ptr->left?ptr->left->data:0;
int right_value=ptr->right?ptr->right->data:0;
return ((ptr->data>left_value)&& (ptr->data<right_value)&&checkbst(ptr->left)&&checkbst(ptr->right));

}
```

• Reply • Share ›

**Sa24** • 5 months ago

```
isBSTUtil(node->left, min, node->data+1) &&
// Allow only distinct values//shouldn't it be +1 as well because head->left <=root ?
isBSTUtil(node->right, node->data+1, max); // Allow only distinct values
}
```

• Reply • Share ›

**daniel** • 5 months ago

i find the method 4 does not work on my case

```
struct bsttree *t1 = create_bst();

assert(is_bstempty(t1));

bst_insert(49, t1);

bst_insert(35, t1);

bst_insert(66, t1);

bst_insert(13, t1);

mirror(t1);

assert(!is_bst(t1));
```

destroy_bst(t1);

}

here the last assertion failed, I checked my code and find the mistake is in the is_bst function, but I don't know where is the problem.

• Reply • Share ›

**Duilio Protti** • 5 months ago

In Method 4, the last check on temp array shouldn't look for ascending order, but instead should look for non-decreasing order (think of duplicates).

• Reply • Share ›

**Eknoor** ➚ Duilio Protti • 2 months ago

You cannot have duplicates in a BST

1 • Reply • Share ›

**Dips** • 5 months ago

METHOD 4(Using In-Order Traversal)

It seems, the time complexity is O(n^2) as it is a recursive calls, correct me if I am wrong?

• Reply • Share ›

**Amit** ➚ Dips • 5 months ago

Nope, you are wrong. It is O(n) because each node is accessed exactly one time. This program is a modification of inorder traversal.

1 • Reply • Share ›

**Haikent** • 5 months ago

bool isbst(node<t> *rm ,int min=INT_MIN,int max=INT_MAX)

{

return(rm && !(rm->data<min||rm->data>max) &&((!rm->left||(rm->left&&isbst(rm->left,min,rm->data-1)))&(!rm->right||(rm->right&&isbst(rm->right,rm->data+1,max)))));

}

• Reply • Share ›

**Haikent** • 5 months ago

int isbst(struct node *tm)
{

return(tm&&(!tm->left||(tm->left&&tm->data>tm->left->data)&&isbst(tm->left))&&(!tm->right||(tm->right&&tm->right->data>tm->data)&&isbst(tm->right)));

}

• Reply • Share ›

**Dimag_se_padhai_karo** • 6 months ago

Method 4 is in fact very correct but the method goes against the principle of using a binary search tree in the first place. Please do not that I am talking with respect to significance of the solution in real world application. Very large datasets cannot be allocated with continous memory locations ,and binary search tree comes in the picture. Restoring elements again in the array is just stupidity and the interviewer will laugh at your face and will tell you to kiss his/her(if lucky) in order to get the job which I think certainly you don't want to do. So just for the sake of different solution , do not ignore simple principles that ought to be followed. These simple principle are the key to real world applications of the conceptual idea in consideration such as binary search tree in above.

2 • Reply • Share ›

**Amazing_boy** ➚ Dimag_se_padhai_karo • 6 months ago

fine, but its not necessary to store them in an array to check if they are in ascending order, a flag and temp var would be enough.

1 • Reply • Share ›

**Santhi** • 7 months ago

Thanks for sharing useful information on pointers. If we
understand correctly we can play with the pointers, otherwise they will play

with us! Anyways, you can get more information on pointer in cpp with examples in the following link.

Pointers in Cpp with examples

• Reply • Share ›

**Mohammad Shahid** • 8 months ago

GeeksforGeeks, or all - Its regarding Method 4.

Can you guys please provide working code Method 4 without using static variable. I tried with following code -

{code}

public static boolean isBST(MyTree root, MyTree pre){

if(root!=null){

if(!isBST(root.getLeft(), pre))

return false;

if(pre!=null&&root.getData()<=pre.getData())

return false;

pre = root;

return isBST(root.getRight(), pre);

see more

• Reply • Share ›

**karunakar** Mohammad Shahid • 7 months ago

int prev=INT_MIN;
int check=isbst(head,&prev);

```
....
....
...
int isbst(struct node *head,int *prev)
{
if(head==NULL) return 1;

if(!isbst(head->left,prev))
return 0;

if(head->data<*prev)
return 0;

*prev=head->data;

return isbst(head->right,prev);

}
```

• Reply • Share ›

**Hello_world** • 8 months ago

awesome guys in recent interview i did similar to logic 1 ..i was not aware that my code is buggy until now.
Thanks guys :D

• Reply • Share ›

**neelabhsingh** • 9 months ago

**@GeeksforGeeks**, Please check may be I am wrong, I changed example, as follows,

```
struct node *root = newNode(3);
root->left = newNode(3);
root->right = newNode(3);
root->left->left = newNode(3);
root->left->right = newNode(3);
```

Suppose, all node of same value, it will still BST, But given code is not giving correct output, please find link,
http://ideone.com/o0ffcO
output: Not BST,

• Reply • Share ›

**phoenix** neelabhsingh • 5 months ago

Acc to def of BST- http://geeksquiz.com/binary-se...

BSt has no duplicate values

• Reply • Share ›

**Noha** • 9 months ago

The accepted answer here is pretty concise/cleaner and is optimal:

http://stackoverflow.com/quest...

1 • Reply • Share ›

**Guest** • 10 months ago

can someone try method 4 with the following

----------(006)-------

--------+----------------+

-------(004)-----------(008)

----+---------+ ----------+------+

(002) ------(005)-------------(009)

• Reply • Share ›

**Nizam** • 10 months ago

Input:
4
/
4

method 4 fails.

Root has data as 4. And one left child which also has 4. Only two nodes in all. This should be a BST but method 4 will fail.

• Reply • Share ›

**Nizam** Nizam • 10 months ago

By the way, can a BST have duplicate values ?

• Reply • Share ›

**guest** Nizam • 10 months ago

I think so

• Reply • Share ›

Load more comments