

AMAZON TEST SERIES
To Help Crack Your SDE Interview

//

Enrol Now


GeeksforGeeks

Construct BST from given preorder traversal | Set 2

Difficulty Level : Hard • Last Updated : 06 Jul, 2021

Given preorder traversal of a binary search tree, construct the BST.

For example, if the given traversal is {10, 5, 1, 7, 40, 50}, then the output should be root of following tree.

10
, \



Start Your Coding Journey Now!

[Login](#)[Register](#)

solution that works in $O(n)$ time.

1. Create an empty stack.
2. Make the first value as root. Push it to the stack.
3. Keep on popping while the stack is not empty and the next value is greater than stack's top value. Make this value as the right child of the last popped node. Push the new node to the stack.
4. If the next value is less than the stack's top value, make this value as the left child of the stack's top node. Push the new node to the stack.
5. Repeat steps 2 and 3 until there are items remaining in pre[].

C++

```
// A O(n) iterative program for construction of BST from preorder traversal
#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, *right;
} node;
```



Start Your Coding Journey Now!

[Login](#)[Register](#)

```
Node** array;
} stack;

// A utility function to create a new tree node
Node* newNode( int data )
{
    Node* temp = new Node();
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// A utility function to create a stack of given capacity
Stack* createStack( int capacity )
{
    Stack* stack = new Stack();
    stack->top = -1;
    stack->capacity = capacity;
    stack->array = new Node*[stack->capacity * sizeof( Node* )];
    return stack;
}

// A utility function to check if stack is full
int isFull( Stack* stack )
{
    return stack->top == stack->capacity - 1;
}

// A utility function to check if stack is empty
int isEmpty( Stack* stack )
```



Start Your Coding Journey Now!

[Login](#)[Register](#)

```
{
    if( isFull( stack ) )
        return;
    stack->array[ ++stack->top ] = item;
}

// A utility function to remove an item from stack
Node* pop( Stack* stack )
{
    if( isEmpty( stack ) )
        return NULL;
    return stack->array[ stack->top-- ];
}

// A utility function to get top node of stack
Node* peek( Stack* stack )
{
    return stack->array[ stack->top ];
}

// The main function that constructs BST from pre[]
Node* constructTree ( int pre[], int size )
{
    // Create a stack of capacity equal to size
    Stack* stack = createStack( size );

    // The first element of pre[] is always root
    Node* root = newNode( pre[0] );

    // Push root
```



Start Your Coding Journey Now!

[Login](#)[Register](#)

```
for ( i = 1; i < size; ++i )
{
    temp = NULL;

    /* Keep on popping while the next value is greater than
    stack's top value. */
    while ( !isEmpty( stack ) && pre[i] > peek( stack )->data )
        temp = pop( stack );

    // Make this greater value as the right child
    // and push it to the stack
    if ( temp != NULL )
    {
        temp->right = newNode( pre[i] );
        push( stack, temp->right );
    }

    // If the next value is less than the stack's top
    // value, make this value as the left child of the
    // stack's top node. Push the new node to stack
    else
    {
        peek( stack )->left = newNode( pre[i] );
        push( stack, peek( stack )->left );
    }
}

return root;
}
```



Start Your Coding Journey Now!

[Login](#)[Register](#)

```
printInorder(node->left);
cout<<node->data<<" ";
printInorder(node->right);
}

// Driver program to test above functions
int main ()
{
    int pre[] = {10, 5, 1, 7, 40, 50};
    int size = sizeof( pre ) / sizeof( pre[0] );

    Node *root = constructTree(pre, size);

    cout<<"Inorder traversal of the constructed tree: \n";
    printInorder(root);

    return 0;
}

//This code is contributed by rathbhupendra
```

C

```
// A O(n) iterative program for construction of BST from preorder traversal
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
```

Start Your Coding Journey Now!

[Login](#)[Register](#)

```
} Node;

// A Stack has array of Nodes, capacity, and top
typedef struct Stack
{
    int top;
    int capacity;
    Node* *array;
} Stack;

// A utility function to create a new tree node
Node* newNode( int data )
{
    Node* temp = (Node *)malloc( sizeof( Node ) );
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// A utility function to create a stack of given capacity
Stack* createStack( int capacity )
{
    Stack* stack = (Stack *)malloc( sizeof( Stack ) );
    stack->top = -1;
    stack->capacity = capacity;
    stack->array = (Node **)malloc( stack->capacity * sizeof( Node* ) );
    return stack;
}

// A utility function to check if stack is full
```



Start Your Coding Journey Now!

[Login](#)[Register](#)

```
int isEmpty( Stack* stack )
{
    return stack->top == -1;
}

// A utility function to push an item to stack
void push( Stack* stack, Node* item )
{
    if( isFull( stack ) )
        return;
    stack->array[ ++stack->top ] = item;
}

// A utility function to remove an item from stack
Node* pop( Stack* stack )
{
    if( isEmpty( stack ) )
        return NULL;
    return stack->array[ stack->top-- ];
}

// A utility function to get top node of stack
Node* peek( Stack* stack )
{
    return stack->array[ stack->top ];
}

// The main function that constructs BST from pre[]
Node* constructTree ( int pre[], int size )
{
```



Start Your Coding Journey Now!

[Login](#)[Register](#)

```
// Push root
push( stack, root );

int i;
Node* temp;

// Iterate through rest of the size-1 items of given preorder array
for ( i = 1; i < size; ++i )
{
    temp = NULL;

    /* Keep on popping while the next value is greater than
       stack's top value. */
    while ( !isEmpty( stack ) && pre[i] > peek( stack )->data )
        temp = pop( stack );

    // Make this greater value as the right child
    // and push it to the stack
    if ( temp != NULL )
    {
        temp->right = newNode( pre[i] );
        push( stack, temp->right );
    }

    // If the next value is less than the stack's top
    // value, make this value as the left child of the
    // stack's top node. Push the new node to stack
    else
    {
        peek( stack )->left = newNode( pre[i] );
```



Start Your Coding Journey Now!

[Login](#)[Register](#)

```
// A utility function to print inorder traversal of a Binary Tree
void printInorder (Node* node)
{
    if (node == NULL)
        return;
    printInorder(node->left);
    printf("%d ", node->data);
    printInorder(node->right);
}

// Driver program to test above functions
int main ()
{
    int pre[] = {10, 5, 1, 7, 40, 50};
    int size = sizeof( pre ) / sizeof( pre[0] );

    Node *root = constructTree(pre, size);

    printf("Inorder traversal of the constructed tree: \n");
    printInorder(root);

    return 0;
}
```

**Java**

Start Your Coding Journey Now!

[Login](#)[Register](#)

```
int data;
Node left, right;

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

class BinaryTree {


    // The main function that constructs BST from pre[]
    Node constructTree(int pre[], int size) {

        // The first element of pre[] is always root
        Node root = new Node(pre[0]);

        Stack<Node> s = new Stack<Node>();

        // Push root
        s.push(root);

        // Iterate through rest of the size-1 items of given preorder array
        for (int i = 1; i < size; ++i) {
            Node temp = null;

            /* Keep on popping while the next value is greater than
            stack's top value. */
            while (!s.isEmpty() && pre[i] > s..data) {
```



Start Your Coding Journey Now!

[Login](#)[Register](#)

```
        temp.right = new Node(pre[i]);
        s.push(temp.right);
    }

    // If the next value is less than the stack's top
    // value, make this value as the left child of the
    // stack's top node. Push the new node to stack
    else {
        temp = s.peek();
        temp.left = new Node(pre[i]);
        s.push(temp.left);
    }
}

return root;
}

// A utility function to print inorder traversal of a Binary Tree
void printInorder(Node node) {
    if (node == null) {
        return;
    }
    printInorder(node.left);
    System.out.print(node.data + " ");
    printInorder(node.right);
}

// Driver program to test above functions
public static void main(String[] args) {
    BinaryTree tree = new BinaryTree();
```



Start Your Coding Journey Now!

[Login](#)[Register](#)

```
}
```

```
// This code has been contributed by Mayank Jaiswal
```

Python3

```
# Python3 program to construct BST
# from given preorder traversal

# A binary tree node
class Node:

    def __init__(self, data = 0):
        self.data = data
        self.left = None
        self.right = None

class BinaryTree :

    # The main function that constructs BST from pre[]
    def constructTree(self, pre, size):

        # The first element of pre[] is always root
        root = Node(pre[0])

        s = []

        # append root
```



Start Your Coding Journey Now!

[Login](#)[Register](#)

```
while ( i < size):
    temp = None

    # Keep on popping while the next value
    # is greater than stack's top value.
    while (len(s) > 0 and pre[i] > s[-1].data):
        temp = s.pop()

    # Make this greater value as the right child
    # and append it to the stack
    if (temp != None):
        temp.right = Node(pre[i])
        s.append(temp.right)

    # If the next value is less than the stack's top
    # value, make this value as the left child of the
    # stack's top node. append the new node to stack
    else :
        temp = s[-1]
        temp.left = Node(pre[i])
        s.append(temp.left)
    i = i + 1
```

```
return root
```

```
# A utility function to print
# inorder traversal of a Binary Tree
def printInorder(self,node):
    if (node == None):
        return
```



Start Your Coding Journey Now!

[Login](#)[Register](#)

```
tree = BinaryTree()
pre = [10, 5, 1, 7, 40, 50]
size = len(pre)
root = tree.constructTree(pre, size)
print("Inorder traversal of the constructed tree is ")
tree.printInorder(root)
```

This code is contributed by Arnab Kundu

C#

```
using System;
using System.Collections.Generic;

// c# program to construct BST from given preorder traversal

// A binary tree node
public class Node
{

    public int data;
    public Node left, right;

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

Start Your Coding Journey Now!

[Login](#)[Register](#)

```
public virtual Node constructTree(int[] pre, int size)
{
    // The first element of pre[] is always root
    Node root = new Node(pre[0]);

    Stack<Node> s = new Stack<Node>();

    // Push root
    s.Push(root);

    // Iterate through rest of the size-1 items of given preorder array
    for (int i = 1; i < size; ++i)
    {
        Node temp = null;

        /* Keep on popping while the next value is greater than
        stack's top value. */
        while (s.Count > 0 && pre[i] > s.Peek().data)
        {
            temp = s.Pop();
        }

        // Make this greater value as the right child
        // and push it to the stack
        if (temp != null)
        {
            temp.right = new Node(pre[i]);
            s.Push(temp.right);
        }
    }
}
```



Start Your Coding Journey Now!

[Login](#)[Register](#)

```
        temp = s.Peek();
        temp.left = new Node(pre[i]);
        s.Push(temp.left);
    }
}

return root;
}

// A utility function to print inorder traversal of a Binary Tree
public virtual void printInorder(Node node)
{
    if (node == null)
    {
        return;
    }
    printInorder(node.left);
    Console.Write(node.data + " ");
    printInorder(node.right);
}

// Driver program to test above functions
public static void Main(string[] args)
{
    BinaryTree tree = new BinaryTree();
    int[] pre = new int[]{10, 5, 1, 7, 40, 50};
    int size = pre.Length;
    Node root = tree.constructTree(pre, size);
    Console.WriteLine("Inorder traversal of the constructed tree is ");
    tree.printInorder(root);
}
```



Start Your Coding Journey Now!

[Login](#)[Register](#)

Javascript

```
<script>

// JavaScript program to construct BST
// from given preorder traversal

// A binary tree node
class Node {
  constructor(d) {
    this.data = d;
    this.left = null;
    this.right = null;
  }
}

class BinaryTree {
  // The main function that constructs BST from pre[]
  constructTree(pre, size) {
    // The first element of pre[] is always root
    var root = new Node(pre[0]);

    var s = [];

    // Push root
    s.push(root);

    // Iterate through rest of the size-1
    // items of given preorder array
```

Start Your Coding Journey Now!

[Login](#)[Register](#)

```
        temp = s.pop();
    }

    // Make this greater value as the right child
    // and push it to the stack
    if (temp != null) {
        temp.right = new Node(pre[i]);
        s.push(temp.right);
    }

    // If the next value is less than the stack's top
    // value, make this value as the left child of the
    // stack's top node. Push the new node to stack
    else {
        temp = s[s.length - 1];
        temp.left = new Node(pre[i]);
        s.push(temp.left);
    }
}

return root;
}

// A utility function to print
// inorder traversal of a Binary Tree
printInorder(node) {
    if (node == null) {
        return;
    }
    this.printInorder(node.left);
```



Start Your Coding Journey Now!

[Login](#)[Register](#)

```
var tree = new BinaryTree();  
var pre = [10, 5, 1, 7, 40, 50];  
var size = pre.length;  
var root = tree.constructTree(pre, size);  
document.write(  
  "Inorder traversal of the constructed tree is <br>");  
tree.printInorder(root);
```

</script>

Output:

```
Inorder traversal of the constructed tree is  
1 5 7 10 40 50
```

Time Complexity: $O(n)$. The complexity looks more from first look. If we take a closer look, we can observe that every item is pushed and popped only once. So at most $2n$ push/pop operations are performed in the main loops of `constructTree()`. Therefore, time complexity is $O(n)$.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Start Your Coding Journey Now!

Login

Register

Previous

Serialize and Deserialize a Binary Tree

Next

Construct BST from given preorder traversal | Set 1

RECOMMENDED ARTICLES

Page : 1 2 3

01 Construct Full Binary Tree using its Preorder traversal and Preorder traversal of its mirror tree
03, Oct 17

05 Find Leftmost and Rightmost node of BST from its given preorder traversal
04, Jul 19

02 Construct BST from given preorder traversal | Set 1
11, Oct 12

06 Construct a special tree from given preorder traversal
28, Jul 12



Number of elements smaller than root using

Start Your Coding Journey Now!

[Login](#)[Register](#)**traversal**

22, Jun 18

traversal

26, Aug 17

Article Contributed By :

**GeeksforGeeks**

Vote for difficulty

Current difficulty : [Hard](#)[Easy](#)[Normal](#)[Medium](#)[Hard](#)[Expert](#)**Improved By :** [shrikanth13](#), [rathbhupendra](#), [andrew1234](#), [itwasme](#)**Article Tags :** [Amazon](#), [Binary Search Tree](#), [Stack](#), [Tree](#)**Practice Tags :** [Amazon](#), [Stack](#), [Binary Search Tree](#), [Tree](#)[Improve Article](#)[Report Issue](#)

Start Your Coding Journey Now!

[Login](#)[Register](#)[Load Comments](#)

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Company

[About Us](#)[Careers](#)[In Media](#)[Contact Us](#)[Privacy Policy](#)[Copyright Policy](#)

Learn

[Algorithms](#)[Data Structures](#)[SDE Cheat Sheet](#)[Machine learning](#)[CS Subjects](#)[Video Tutorials](#)

News

[Top News](#)[Technology](#)[Work & Career](#)[Business](#)[Finance](#)[Lifestyle](#)

Languages

[Python](#)[Java](#)[CPP](#)[Golang](#)[C#](#)[SQL](#)

Web Development

[Web Tutorials](#)[Django Tutorial](#)[HTML](#)[CSS](#)[JavaScript](#)[Bootstrap](#)

Contribute

[Write an Article](#)[Improve an Article](#)[Pick Topics to Write](#)[Write Interview Experience](#)[Internships](#)[Video Internship](#)

@geeksforgeeks  All rights reserved