



[Array](#) [Matrix](#) [Strings](#) [Hashing](#) [Linked List](#) [Stack](#) [Queue](#) [Binary Tree](#) [Binary Search Tree](#) [Heap](#) [Graph](#) [Searching](#) [Sorting](#)

**DSA Self-Paced  
Course**

- ✓ Curated by experts
- ✓ Trusted by 1 Lac+ students.

**Enrol Now**

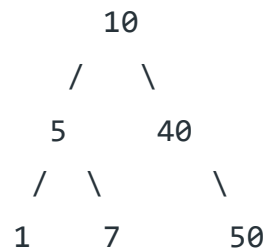


# Construct BST from given preorder traversal | Set 1

Difficulty Level : Hard • Last Updated : 09 Mar, 2022

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 the root of the following tree.



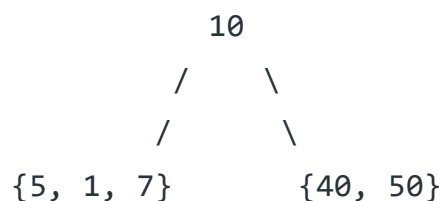
## Start Your Coding Journey Now!

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

### Method 1 (O(n<sup>2</sup>) time complexity)

The first element of preorder traversal is always root. We first construct the root. Then we find the index of the first element which is greater than the root. Let the index be 'i'. The values between root and 'i' will be part of the left subtree, and the values between 'i' (inclusive) and 'n-1' will be part of the right subtree. Divide given pre[] at index "i" and recur for left and right sub-trees.

**For example** in {10, 5, 1, 7, 40, 50}, 10 is the first element, so we make it root. Now we look for the first element greater than 10, we find 40. So we know the structure of BST is as following.



We recursively follow above steps for subarrays {5, 1, 7} and {40, 50}, and get the complete tree.

### C++

```

/* A O(n^2) 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
  
```

## Start Your Coding Journey Now!

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

```
};

// A utility function to create a node
node* newNode(int data)
{
    node* temp = new node();

    temp->data = data;
    temp->left = temp->right = NULL;

    return temp;
}

// A recursive function to construct Full from pre[].
// preIndex is used to keep track of index in pre[].
node* constructTreeUtil(int pre[], int* preIndex, int low,
                        int high, int size)
{
    // Base case
    if (*preIndex >= size || low > high)
        return NULL;

    // The first node in preorder traversal is root. So take
    // the node at preIndex from pre[] and make it root, and
    // increment preIndex
    node* root = newNode(pre[*preIndex]);
    *preIndex = *preIndex + 1;

    // If the current subarray has only one element, no need
    // to recur
```



## Start Your Coding Journey Now!

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

```
    if (pre[i] > root->data)
        break;

    // Use the index of element found in preorder to divide
    // preorder array in two parts. Left subtree and right
    // subtree
    root->left = constructTreeUtil(pre, preIndex, *preIndex,
                                  i - 1, size);

    root->right
        = constructTreeUtil(pre, preIndex, i, high, size);

    return root;
}

// The main function to construct BST from given preorder
// traversal. This function mainly uses constructTreeUtil()
node* constructTree(int pre[], int size)
{
    int preIndex = 0;
    return constructTreeUtil(pre, &preIndex, 0, size - 1,
                              size);
}

// A utility function to print inorder traversal of a Binary
// Tree
void printInorder(node* node)
{
    if (node == NULL)
        return;
    printInorder(node->left);
```



## Start Your Coding Journey Now!

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

```
{  
    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^2) program for construction of BST from preorder  
 * traversal */  
#include <stdio.h>  
#include <stdlib.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;  
};
```

## Start Your Coding Journey Now!

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

```
temp->data = data;
temp->left = temp->right = NULL;

return temp;
}

// A recursive function to construct Full from pre[].
// preIndex is used to keep track of index in pre[].
struct node* constructTreeUtil(int pre[], int* preIndex,
                              int low, int high, int size)
{
    // Base case
    if (*preIndex >= size || low > high)
        return NULL;

    // The first node in preorder traversal is root. So take
    // the node at preIndex from pre[] and make it root, and
    // increment preIndex
    struct node* root = newNode(pre[*preIndex]);
    *preIndex = *preIndex + 1;

    // If the current subarray has only one element, no need
    // to recur
    if (low == high)
        return root;

    // Search for the first element greater than root
    int i;
    for (i = low; i <= high; ++i)
```



## Start Your Coding Journey Now!

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

```
root->left = constructTreeUtil(pre, preIndex, *preIndex,
                              i - 1, size);

root->right
    = constructTreeUtil(pre, preIndex, i, high, size);

return root;
}
```

```
// The main function to construct BST from given preorder
// traversal. This function mainly uses constructTreeUtil()
struct node* constructTree(int pre[], int size)
{
    int preIndex = 0;
    return constructTreeUtil(pre, &preIndex, 0, size - 1,
                             size);
}
```

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

```
// Driver code
int main()
```



## Start Your Coding Journey Now!

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

```
printf("Inorder traversal of the constructed tree: \n");  
printInorder(root);  
  
return 0;  
}
```

### Java

```
// Java program to construct BST from given preorder  
// traversal  
  
// A binary tree node  
class Node {  
  
    int data;  
    Node left, right;  
  
    Node(int d)  
    {  
        data = d;  
        left = right = null;  
    }  
}  
  
class Index {  
  
    int index = 0;  
}
```



## Start Your Coding Journey Now!

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

```
// preIndex is used to keep track of index in pre[].
Node constructTreeUtil(int pre[], Index preIndex,
                      int low, int high, int size)
{
    // Base case
    if (preIndex.index >= size || low > high) {
        return null;
    }

    // The first node in preorder traversal is root. So
    // take the node at preIndex from pre[] and make it
    // root, and increment preIndex
    Node root = new Node(pre[preIndex.index]);
    preIndex.index = preIndex.index + 1;

    // If the current subarray has only one element, no
    // need to recur
    if (low == high) {
        return root;
    }

    // Search for the first element greater than root
    int i;
    for (i = low; i <= high; ++i) {
        if (pre[i] > root.data) {
            break;
        }
    }
}
```



## Start Your Coding Journey Now!

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

```
        high, size);

    return root;
}

// The main function to construct BST from given
// preorder traversal. This function mainly uses
// constructTreeUtil()
Node constructTree(int pre[], int size)
{
    return constructTreeUtil(pre, index, 0, size - 1,
                             size);
}

// 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 code
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

```
 # A O(n^2) Python3 program for  
 # construction of BST from preorder traversal  
 # A binary tree node  
 class Node():  
  
    # A constructor to create a new node  
    def __init__(self, data):  
        self.data = data  
        self.left = None  
        self.right = None  
  
# constructTreeUtil.preIndex is a static variable of  
# function constructTreeUtil  
  
# Function to get the value of static variable  
# constructTreeUtil.preIndex  
def getPreIndex():
```



## Start Your Coding Journey Now!

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

```
def incrementPreIndex():
    constructTreeUtil.preIndex += 1

# A recursive function to construct Full from pre[].
# preIndex is used to keep track of index in pre[].

def constructTreeUtil(pre, low, high):

    # Base Case
    if(low > high):
        return None

    # The first node in preorder traversal is root. So take
    # the node at preIndex from pre[] and make it root,
    # and increment preIndex
    root = Node(pre[getPreIndex()])
    incrementPreIndex()

    # If the current subarray has only one element,
    # no need to recur
    if low == high:
        return root

    r_root = -1

    # Search for the first element greater than root
    for i in range(low, high+1):
        if (pre[i] > root.data):
            r_root = i
```



## Start Your Coding Journey Now!

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

```
    r_root = getPreIndex() + (high - low)

    # Use the index of element found in preorder to divide
    # preorder array in two parts. Left subtree and right
    # subtree
    root.left = constructTreeUtil(pre, getPreIndex(), r_root-1)

    root.right = constructTreeUtil(pre, r_root, high)

    return root

# The main function to construct BST from given preorder
# traversal. This function mainly uses constructTreeUtil()

def constructTree(pre):
    size = len(pre)
    constructTreeUtil.preIndex = 0
    return constructTreeUtil(pre, 0, size-1)

def printInorder(root):
    if root is None:
        return
    printInorder(root.left)
    print (root.data,end=' ')
    printInorder(root.right)

# Driver code
```



## Start Your Coding Journey Now!

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

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

### C#

```
using System;

// 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;
    }
}

public class Index {

    public int index = 0;
}

public class BinaryTree {

    public Index index = new Index();
```

## Start Your Coding Journey Now!

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

```
int size)
{
    // Base case
    if (preIndex.index >= size || low > high) {
        return null;
    }

    // The first node in preorder traversal is root. So
    // take the node at preIndex from pre[] and make it
    // root, and increment preIndex
    Node root = new Node(pre[preIndex.index]);
    preIndex.index = preIndex.index + 1;

    // If the current subarray has only one element, no
    // need to recur
    if (low == high) {
        return root;
    }

    // Search for the first element greater than root
    int i;
    for (i = low; i <= high; ++i) {
        if (pre[i] > root.data) {
            break;
        }
    }

    // Use the index of element found in preorder to
    // divide preorder array in two parts. Left subtree
```



## Start Your Coding Journey Now!

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

```
        return root;
    }

    // The main function to construct BST from given
    // preorder traversal. This function mainly uses
    // constructTreeUtil()
    public virtual Node constructTree(int[] pre, int size)
    {
        return constructTreeUtil(pre, index, 0, size - 1,
                                size);
    }

    // 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 code
    public static void Main(string[] args)
    {
        BinaryTree tree = new BinaryTree();
        int[] pre = new int[] { 10, 5, 1, 7, 40, 50 };
        int size = pre.Length;
```





# Start Your Coding Journey Now!

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

```
// This code is contributed by Shrikant13
```

## Output

Inorder traversal of the constructed tree:

```
1 5 7 10 40 50
```

**Time Complexity:**  $O(n^2)$

## Method 2 ( $O(n)$ time complexity )

The idea used here is inspired by method 3 of [this](#) post. The trick is to set a range {min .. max} for every node. Initialize the range as {INT\_MIN .. INT\_MAX}. The first node will definitely be in range, so create a root node. To construct the left subtree, set the range as {INT\_MIN ...root->data}. If a value is in the range {INT\_MIN .. root->data}, the values are part of the left subtree. To construct the right subtree, set the range as {root->data..max .. INT\_MAX}.

Below is the implementation of the above idea:

## C++

```
/* A O(n) program for construction  
of BST from preorder traversal */
```

## Start Your Coding Journey Now!

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

```
public:
    int data;
    node* left;
    node* right;
};

// A utility function to create a node
node* newNode(int data)
{
    node* temp = new node();

    temp->data = data;
    temp->left = temp->right = NULL;

    return temp;
}

// A recursive function to construct
// BST from pre[]. preIndex is used
// to keep track of index in pre[].
node* constructTreeUtil(int pre[], int* preIndex, int key,
                        int min, int max, int size)
{
    // Base case
    if (*preIndex >= size)
        return NULL;

    node* root = NULL;

    // If current element of pre[] is in range, then
```



# Start Your Coding Journey Now!

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

```
if (*preIndex < size) {
    // Construct the subtree under root
    // All nodes which are in range
    // {min .. key} will go in left
    // subtree, and first such node
    // will be root of left subtree.
    root->left = constructTreeUtil(pre, preIndex,
                                   pre[*preIndex],
                                   min, key, size);
}
if (*preIndex < size) {
    // All nodes which are in range
    // {key..max} will go in right
    // subtree, and first such node
    // will be root of right subtree.
    root->right = constructTreeUtil(pre, preIndex,
                                    pre[*preIndex],
                                    key, max, size);
}
}

return root;
}

// The main function to construct BST
// from given preorder traversal.
// This function mainly uses constructTreeUtil()
node* constructTree(int pre[], int size)
{
```



## Start Your Coding Journey Now!

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

```
// traversal of a Binary Tree
void printInorder(node* node)
{
    if (node == NULL)
        return;
    printInorder(node->left);
    cout << node->data << " ";
    printInorder(node->right);
}

// Driver code
int main()
{
    int pre[] = { 10, 5, 1, 7, 40, 50 };
    int size = sizeof(pre) / sizeof(pre[0]);

    // Function call
    node* root = constructTree(pre, size);

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

    return 0;
}

// This code is contributed by rathbhupendra
```

**C**

## Start Your Coding Journey Now!

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

```
/* 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;
};

// A utility function to create a node
struct node* newNode(int data)
{
    struct node* temp
        = (struct node*)malloc(sizeof(struct node));

    temp->data = data;
    temp->left = temp->right = NULL;

    return temp;
}

// A recursive function to construct BST from pre[].
// preIndex is used to keep track of index in pre[].
struct node* constructTreeUtil(int pre[], int* preIndex,
                                int key, int min, int max,
                                int size)
{
    // Base case
    if (*preIndex >= size)
        return NULL;
```



## Start Your Coding Journey Now!

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

```
// increment *preIndex
root = newNode(key);
*preIndex = *preIndex + 1;

if (*preIndex < size) {
    // Construct the subtree under root
    // All nodes which are in range {min .. key}
    // will go in left subtree, and first such node
    // will be root of left subtree.
    root->left = constructTreeUtil(pre, preIndex,
                                  pre[*preIndex],
                                  min, key, size);
}
if (*preIndex < size) {
    // All nodes which are in range {key..max} will
    // go in right subtree, and first such node will
    // be root of right subtree.
    root->right = constructTreeUtil(pre, preIndex,
                                    pre[*preIndex],
                                    key, max, size);
}

return root;
}

// The main function to construct BST from given preorder
// traversal. This function mainly uses constructTreeUtil()
struct node* constructTree(int pre[], int size)
{
```



## Start Your Coding Journey Now!

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

```
// Tree
void printInorder(struct node* node)
{
    if (node == NULL)
        return;
    printInorder(node->left);
    printf("%d ", node->data);
    printInorder(node->right);
}

// Driver code
int main()
{
    int pre[] = { 10, 5, 1, 7, 40, 50 };
    int size = sizeof(pre) / sizeof(pre[0]);

    // function call
    struct node* root = constructTree(pre, size);

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

    return 0;
}
```



### Java



// Java program to construct BST from given preorder

## Start Your Coding Journey Now!

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

```
Node left, right;

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

class Index {

    int index = 0;
}

class BinaryTree {

    Index index = new Index();

    // A recursive function to construct BST from pre[].
    // preIndex is used to keep track of index in pre[].
    Node constructTreeUtil(int pre[], Index preIndex,
                           int key, int min, int max,
                           int size)

    {

        // Base case
        if (preIndex.index >= size) {
            return null;
        }
```





## Start Your Coding Journey Now!

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

```
// Allocate memory for root of this
// subtree and increment *preIndex
root = new Node(key);
preIndex.index = preIndex.index + 1;

if (preIndex.index < size) {

    // Construct the subtree under root
    // All nodes which are in range {min .. key}
    // will go in left subtree, and first such
    // node will be root of left subtree.
    root.left = constructTreeUtil(
        pre, preIndex, pre[preIndex.index], min,
        key, size);
}
if (preIndex.index < size) {
    // All nodes which are in range {key..max}
    // will go in right subtree, and first such
    // node will be root of right subtree.
    root.right = constructTreeUtil(
        pre, preIndex, pre[preIndex.index], key,
        max, size);
}
}

return root;
}

// The main function to construct BST from given
// preorder traversal. This function mainly uses
```



# Start Your Coding Journey Now!

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

```
Integer.MAX_VALUE, size);  
}  
  
// 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 code  
public static void main(String[] args)  
{  
    BinaryTree tree = new BinaryTree();  
    int pre[] = new int[] { 10, 5, 1, 7, 40, 50 };  
    int size = pre.length;  
  
    // Function call  
    Node root = tree.constructTree(pre, size);  
    System.out.println(  
        "Inorder traversal of the constructed tree is ");  
    tree.printInorder(root);  
}  
}
```



## Start Your Coding Journey Now!

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

# A O(n) program for construction of BST from preorder traversal

INT\_MIN = -float("inf")

INT\_MAX = float("inf")

# A Binary tree node

**class** Node:

# Constructor to created a new node

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

self.data = data

self.left = None

self.right = None

# Methods to get and set the value of static variable

# constructTreeUtil.preIndex for function construcTreeUtil()

**def** getPreIndex():

**return** constructTreeUtil.preIndex

**def** incrementPreIndex():

constructTreeUtil.preIndex += 1

# A recursive function to construct BST from pre[].

# preIndex is used to keep track of index in pre[]

## Start Your Coding Journey Now!

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

```
root = None
```

```
# If current element of pre[] is in range, then  
# only it is part of current subtree
```

```
if(key > mini and key < maxi):
```

```
    # Allocate memory for root of this subtree  
    # and increment constructTreeUtil.preIndex
```

```
    root = Node(key)  
    incrementPreIndex()
```

```
if(getPreIndex() < size):
```

```
    # Construct the subtree under root  
    # All nodes which are in range {min.. key} will  
    # go in left subtree, and first such node will  
    # be root of left subtree  
    root.left = constructTreeUtil(pre,  
                                  pre[getPreIndex()],  
                                  mini, key, size)
```

```
if(getPreIndex() < size):
```

```
    # All nodes which are in range{key..max} will  
    # go to right subtree, and first such node will  
    # be root of right subtree  
    root.right = constructTreeUtil(pre,  
                                    pre[getPreIndex()],  
                                    key, maxi, size)
```



## Start Your Coding Journey Now!

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

```
def constructTree(pre):
    constructTreeUtil.preIndex = 0
    size = len(pre)
    return constructTreeUtil(pre, pre[0], INT_MIN, INT_MAX, size)

# A utility function to print inorder traversal of Binary Tree
def printInorder(node):

    if node is None:
        return
    printInorder(node.left)
    print (node.data,end=" ")
    printInorder(node.right)

# Driver code
pre = [10, 5, 1, 7, 40, 50]

# Function call
root = constructTree(pre)

print ("Inorder traversal of the constructed tree: ")
printInorder(root)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```



# Start Your Coding Journey Now!

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

```
// A binary tree node
```

```
public class Node {
```

```
    public int data;
```

```
    public Node left, right;
```

```
    public Node(int d)
```

```
    {
```

```
        data = d;
```

```
        left = right = null;
```

```
    }
```

```
}
```

```
public class Index {
```

```
    public int index = 0;
```

```
}
```

```
public class BinaryTree {
```

```
    public Index index = new Index();
```

```
    // A recursive function to construct BST from pre[].
```

```
    // preIndex is used to keep track of index in pre[].
```

```
    public virtual Node constructTreeUtil(int[] pre,
                                         Index preIndex,
                                         int key, int min,
                                         int max, int size)
```

```
    {
```

```
        // Base case
```

## Start Your Coding Journey Now!

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

```
// If current element of pre[] is in range, then
// only it is part of current subtree
if (key > min && key < max) {

    // Allocate memory for root of this subtree
    // and increment *preIndex
    root = new Node(key);
    preIndex.index = preIndex.index + 1;

    if (preIndex.index < size) {

        // Construct the subtree under root
        // All nodes which are in range
        // {min .. key} will go in left
        // subtree, and first such node will
        // be root of left subtree.
        root.left = constructTreeUtil(
            pre, preIndex, pre[preIndex.index], min,
            key, size);
    }
    if (preIndex.index < size) {
        // All nodes which are in range
        // {key..max} will go in right
        // subtree, and first such node
        // will be root of right subtree.
        root.right = constructTreeUtil(
            pre, preIndex, pre[preIndex.index], key,
            max, size);
    }
}
```



# Start Your Coding Journey Now!

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

```
// constructTreeUtil()
public virtual Node constructTree(int[] pre, int size)
{
    return constructTreeUtil(pre, index, pre[0],
                              int.MinValue, int.MaxValue,
                              size);
}

// 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 code
public static void Main(string[] args)
{
    BinaryTree tree = new BinaryTree();
    int[] pre = new int[] { 10, 5, 1, 7, 40, 50 };
    int size = pre.Length;

    // Function call
    Node root = tree.constructTree(pre, size);
}
```





# Start Your Coding Journey Now!

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

// This code is contributed by Shrikant13

## Javascript

```
<script>
// javascript program to construct BST from given preorder
// traversal

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

class Index {
    constructor(){
        this.index = 0;
    }
}

index = new Index();

// A recursive function to construct BST from pre.
// preIndex is used to keep track of index in pre.
function constructTreeUtil(pre, preIndex , key , min , max , size) {
```

## Start Your Coding Journey Now!

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

```
var root = null;

// If current element of pre is in range, then
// only it is part of current subtree
if (key > min && key < max) {

    // Allocate memory for root of this
    // subtree and increment *preIndex
    root = new Node(key);
    preIndex.index = preIndex.index + 1;

    if (preIndex.index < size) {

        // Construct the subtree under root
        // All nodes which are in range {min .. key}
        // will go in left subtree, and first such
        // node will be root of left subtree.
        root.left = constructTreeUtil(pre, preIndex,
        pre[preIndex.index], min, key, size);
    }
    if (preIndex.index < size)
    {

        // All nodes which are in range {key..max}
        // will go in right subtree, and first such
        // node will be root of right subtree.
        root.right = constructTreeUtil(pre, preIndex,
        pre[preIndex.index], key, max, size);
    }
}
```



## Start Your Coding Journey Now!

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

```
// constructTreeUtil()
function constructTree(pre , size) {
    var preIndex = 0;
    return constructTreeUtil(pre, index, pre[0],
        Number.MIN_VALUE, Number.MAX_VALUE, size);
}

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

// Driver code
var pre = [ 10, 5, 1, 7, 40, 50 ];
var size = pre.length;

// Function call
var root = constructTree(pre, size);
document.write("Inorder traversal of the constructed tree is <br/>");
printInorder(root);

// This code is contributed by Rajput-Ji
</script>
```



## Start Your Coding Journey Now!

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

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

We will soon publish a  $O(n)$  iterative solution as a separate post.

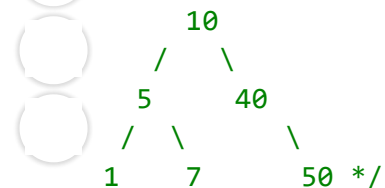
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

**Method 3 (  $O(n^2)$  time complexity ):**

Simply do that just by using the recursion concept and iterating through the array of the given elements like below.

### Java

```
/*Construct a BST from given pre-order traversal
for example if the given traversal is {10, 5, 1, 7, 40, 50},
then the output should be the root of the following tree.
```



```
class Node {
    int data;
    Node left, right;
    Node(int data)
    {
        this.data = data;
```

# Start Your Coding Journey Now!

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

```
// This will create the BST
public static Node createNode(Node node, int data)
{
    if (node == null)
        node = new Node(data);

    if (node.data > data)
        node.left = createNode(node.left, data);
    if (node.data < data)
        node.right = createNode(node.right, data);

    return node;
}

// A wrapper function of createNode
public static void create(int data)
{
    node = createNode(node, data);
}

// A function to print BST in inorder
public static void inorderRec(Node root)
{
    if (root != null) {
        inorderRec(root.left);
        System.out.println(root.data);
        inorderRec(root.right);
    }
}
```



## Start Your Coding Journey Now!

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

```

        create(nodeData[i]);
    }
    inorderRec(node);
}
}

```

### C#

/\*Construct a BST from given pre-order traversal  
for example if the given traversal is {10, 5, 1, 7, 40, 50},  
then the output should be the root of the following tree.

```

      10
     /  \
    5    40
   /  \  / \
  1   7 50 */

```

```

using System;
public class Node {
    public int data;
    public Node left, right;
    public Node(int data)
    {
        this.data = data;
        this.left = this.right = null;
    }
}

```

```

public class CreateBSTFromPreorder {

```

## Start Your Coding Journey Now!

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

```
        node = new Node(data);

        if (node.data > data)
            node.left = createNode(node.left, data);
        if (node.data < data)
            node.right = createNode(node.right, data);

        return node;
    }

    // A wrapper function of createNode
    public static void create(int data)
    {
        node = createNode(node, data);
    }

    // A function to print BST in inorder
    public static void inorderRec(Node root)
    {
        if (root != null) {
            inorderRec(root.left);
            Console.WriteLine(root.data);
            inorderRec(root.right);
        }
    }

    // Driver Code
    public static void Main(String[] args)
    {
        int[] nodeData = { 10, 5, 1, 7, 40, 50 };
    }
```



# Start Your Coding Journey Now!

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

// This code is contributed by Rajput-Ji

## Javascript

```
<script>
/*Construct a BST from given pre-order traversal
for example if the given traversal is {10, 5, 1, 7, 40, 50},
then the output should be the root of the following tree.
```

```
    10
   /  \
  5    40
 /  \  \
1   7  50 */
```

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

```
var node;
```

```
// This will create the BST
function createNode(node , data) {
  if (node == null)
    node = new Node(data);
```



## Start Your Coding Journey Now!

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

```
    return node;
}

// A wrapper function of createNode
function create(data) {
    node = createNode(node, data);
}

// A function to print BST in inorder
function inorderRec(root) {
    if (root != null) {
        inorderRec(root.left);
        document.write(root.data+"<br/>");
        inorderRec(root.right);
    }
}

// Driver Code
var nodeData = [ 10, 5, 1, 7, 40, 50 ];
for (i = 0; i < nodeData.length; i++)
{
    create(nodeData[i]);
}
inorderRec(node);
```

```
// This code is contributed by Rajput-Ji
</script>
```



## Start Your Coding Journey Now!

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

10

40

50

**Like** 193[Previous](#)[Threaded Binary Search Tree | Deletion](#)[Next](#)[How to handle duplicates in Binary Search Tree?](#)

## Start Your Coding Journey Now!

[Login](#)[Register](#)**Set 2**

11, Oct 12

**traversal**

02, Oct 17

**02** Construct BST from given preorder traversal | Set 3 (Naive Method)  
27, Apr 20

**06** Construct a BST from given postorder traversal using Stack  
28, Oct 18

**03** Find postorder traversal of BST from preorder traversal  
22, Jun 18

**07** Check if a given array can represent Preorder Traversal of Binary Search Tree  
30, Oct 15

**04** Number of elements smaller than root using preorder traversal of a BST  
31, Aug 18

**08** Two nodes of a BST are swapped, correct the BST | Set-2  
12, Jun 19

### Article Contributed By :

**GeeksforGeeks**

### Vote for difficulty

Current difficulty : [Hard](#)[Easy](#)[Normal](#)[Medium](#)[Hard](#)[Expert](#)

## Start Your Coding Journey Now!

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

Article Tags : [Binary Search Tree](#)

Practice Tags : [Binary Search Tree](#)

[Improve Article](#)[Report Issue](#)

Writing code in comment? Please use [ide.geeksforgeeks.org](https://ide.geeksforgeeks.org), generate link and share the link here.

[Load Comments](#)

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

[feedback@geeksforgeeks.org](mailto:feedback@geeksforgeeks.org)



## Start Your Coding Journey Now!

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

In Media	SDE Cheat Sheet	Technology	CPP	HTML	Pick Topics to Write
Contact Us	Machine learning	Work & Career	Golang	CSS	Write Interview Experience
Privacy Policy	CS Subjects	Business	C#	JavaScript	Internships
Copyright Policy	Video Tutorials	Finance Lifestyle	SQL	Bootstrap	Video Internship

@geeksforgeeks , Some rights reserved

