Sorted Array to Balanced BST

Difficulty Level : Easy • Last Updated : 20 Jan, 2022

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

Examples:

```
Input:  Array {1, 2, 3}
Output: A Balanced BST
     2
   /  \
  1    3

Input: Array {1, 2, 3, 4}
Output: A Balanced BST
      3
    /  \
```

Algorithm

In the previous post, we discussed construction of BST from sorted Linked List. Constructing from sorted array in O(n) time is simpler as we can get the middle element in O(1) time. Following is a simple algorithm where we first find the middle node of list and make it root of the tree to be constructed.

```
1) Get the Middle of the array and make it root.
2) Recursively do same for left half and right half.
      a) Get the middle of left half and make it left child of the root
          created in step 1.
      b) Get the middle of right half and make it right child of the
          root created in step 1.
```

Following is the implementation of the above algorithm. The main code which creates Balanced BST is highlighted.

```
using namespace std;

/* A Binary Tree node */
class TNode
{
    public:
    int data;
    TNode* left;
    TNode* right;
};

TNode* newNode(int data);

/* A function that constructs Balanced
Binary Search Tree from a sorted array */
TNode* sortedArrayToBST(int arr[],
                        int start, int end)
{
    /* Base Case */
    if (start > end)
    return NULL;

    /* Get the middle element and make it root */
    int mid = (start + end)/2;
    TNode *root = newNode(arr[mid]);

    /* Recursively construct the left subtree
    and make it left child of root */
    root->left = sortedArrayToBST(arr, start,
                                  mid - 1);
```

```
/* Helper function that allocates a new node
with the given data and NULL left and right
pointers. */
TNode* newNode(int data)
{
    TNode* node = new TNode();
    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return node;
}

/* A utility function to print
preorder traversal of BST */
void preOrder(TNode* node)
{
    if (node == NULL)
        return;
    cout << node->data << " ";
    preOrder(node->left);
    preOrder(node->right);
}

// Driver Code
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);
```

```
    return 0;
}

// This code is contributed by rathbhupendra
```

C

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

/* A Binary Tree node */
struct TNode
{
    int data;
    struct TNode* left;
    struct TNode* right;
};

struct TNode* newNode(int data);

/* A function that constructs Balanced Binary Search Tree from a sorted array */
struct TNode* sortedArrayToBST(int arr[], int start, int end)
{
    /* Base Case */
    if (start > end)
      return NULL;

    /* Get the middle element and make it root */
```

```
    /* Recursively construct the right subtree and make it
       right child of root */
    root->right = sortedArrayToBST(arr, mid+1, end);

    return root;
}

/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct TNode* newNode(int data)
{
    struct TNode* node = (struct TNode*)
                         malloc(sizeof(struct TNode));
    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return node;
}

/* A utility function to print preorder traversal of BST */
void preOrder(struct TNode* node)
{
    if (node == NULL)
        return;
    printf("%d ", node->data);
    preOrder(node->left);
    preOrder(node->right);
}
```

```
    /* Convert List to BST */
    struct TNode *root = sortedArrayToBST(arr, 0, n-1);
    printf("n PreOrder Traversal of constructed BST ");
    preOrder(root);

    return 0;
}
```

Java

```
// Java program to print BST in given range

// A binary tree node
class Node {

    int data;
    Node left, right;

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

class BinaryTree {

    static Node root;
```

```
    if (start > end) {
        return null;
    }

    /* Get the middle element and make it root */
    int mid = (start + end) / 2;
    Node node = new Node(arr[mid]);

    /* Recursively construct the left subtree and make it
     left child of root */
    node.left = sortedArrayToBST(arr, start, mid - 1);

    /* Recursively construct the right subtree and make it
     right child of root */
    node.right = sortedArrayToBST(arr, mid + 1, end);

    return node;
}

/* A utility function to print preorder traversal of BST */
void preOrder(Node node) {
    if (node == null) {
        return;
    }
    System.out.print(node.data + " ");
    preOrder(node.left);
    preOrder(node.right);
}

public static void main(String[] args) {
```

```
    }
}

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

Python

```
# 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
    mid = (len(arr)) / 2
```

```
    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 ",
```

C#

```
using System;

// C# program to print BST in given range

// 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 BinaryTree
{

    public static Node root;

    /* A function that constructs Balanced Binary Search Tree
     from a sorted array */
    public virtual Node sortedArrayToBST(int[] arr, int start, int end)
    {
```

```
    /* Get the middle element and make it root */
    int mid = (start + end) / 2;
    Node node = new Node(arr[mid]);

    /* Recursively construct the left subtree and make it
     left child of root */
    node.left = sortedArrayToBST(arr, start, mid - 1);

    /* Recursively construct the right subtree and make it
     right child of root */
    node.right = sortedArrayToBST(arr, mid + 1, end);

    return node;
}

/* A utility function to print preorder traversal of BST */
public virtual void preOrder(Node node)
{
    if (node == null)
    {
        return;
    }
    Console.Write(node.data + " ");
    preOrder(node.left);
    preOrder(node.right);
}

public static void Main(string[] args)
{
    BinaryTree tree = new BinaryTree();
```

```
}

  // This code is contributed by Shrikant13
```

Javascript


```
<script>


// JavaScript program to print BST in given range

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

var root = null;

/* A function that constructs Balanced Binary Search Tree
 from a sorted array */
function sortedArrayToBST(arr, start, end)
{
```

```
    var mid = parseInt((start + end) / 2);
    var node = new Node(arr[mid]);
    /* Recursively construct the left subtree and make it
     left child of root */
    node.left = sortedArrayToBST(arr, start, mid - 1);
    /* Recursively construct the right subtree and make it
     right child of root */
    node.right = sortedArrayToBST(arr, mid + 1, end);
    return node;
}
/* A utility function to print preorder traversal of BST */
function preOrder(node)
{
    if (node == null)
    {
        return;
    }
    document.write(node.data + " ");
    preOrder(node.left);
    preOrder(node.right);
}


var arr = [1, 2, 3, 4, 5, 6, 7];
var n = arr.length;
root = sortedArrayToBST(arr, 0, n - 1);
document.write("Preorder traversal of constructed BST<br>");
preOrder(root);

</script>
```

```
4 2 1 3 6 5 7
```

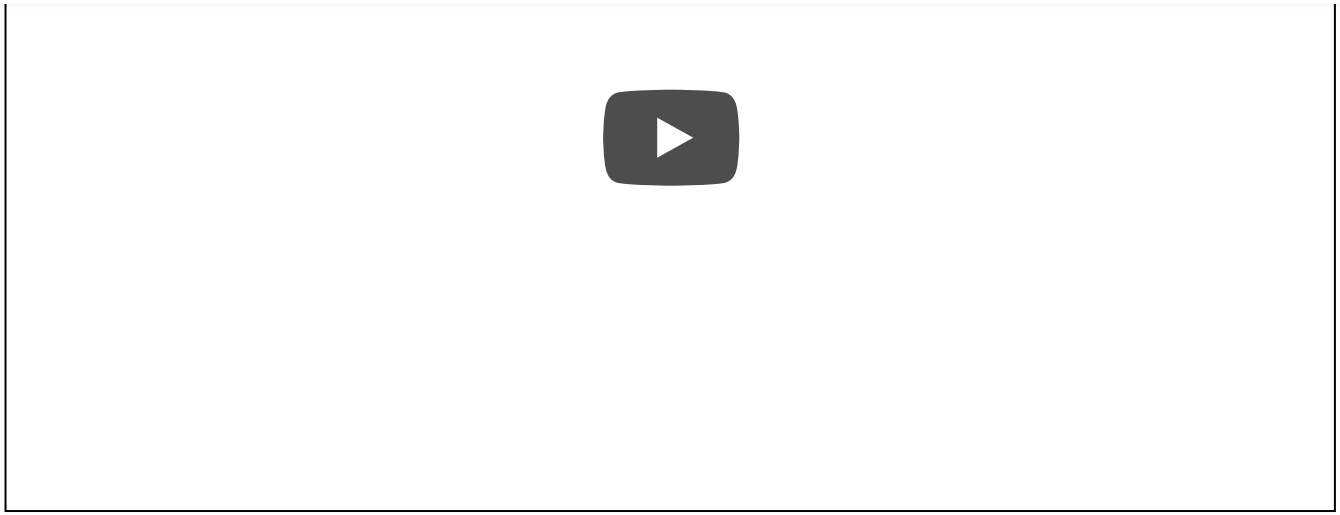
```
Tree representation of above output:
     4
 2       6
1  3   5   7
```

Time Complexity: O(n)

Following is the recurrence relation for sortedArrayToBST().

```
T(n) = 2T(n/2) + C
T(n) -->  Time taken for an array of size n
 C   -->  Constant (Finding middle of array and linking root to left
                    and right subtrees take constant time)
```


The above recurrence can be solved using Master Theorem as it falls in case 1.



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

Like 90

Previous

Sorted Linked List to Balanced BST

Next

Find the node with minimum value in a Binary Search Tree

02 **Sorted Linked List to Balanced BST**
17, Jan 12

03 **Comparison between Height Balanced Tree and Weight Balanced Tree**
27, Dec 21

04 **Find if there is a triplet in a Balanced BST that adds to zero**
09, Mar 13

06 **Split a BST into two balanced BSTs based on a value K**
19, Apr 20

07 **Check if the Binary Tree contains a balanced BST of size K**
06, Jul 20

08 **Create a balanced BST using vector in C++ STL**
26, Dec 21

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Easy

Easy | Normal | Medium | Hard | Expert

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

feedback@geeksforgeeks.org