Sorted Array to Balanced BST

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
    /  \
   2    4
 /
1
```

We strongly recommend that you click here and practice it, before moving on to the solution.

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.

C

```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 */
    int mid = (start + end)/2;
    struct TNode *root = newNode(arr[mid]);

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

    /* 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);
}

/* Driver program to test above functions */
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6, 7};
    int n = sizeof(arr)/sizeof(arr[0]);

    /* 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;

    /* A function that constructs Balanced Binary Search Tree
     from a sorted array */
    Node 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;
        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) {
        BinaryTree tree = new BinaryTree();
        int arr[] = new int[]{1, 2, 3, 4, 5, 6, 7};
        int n = arr.length;
        root = tree.sortedArrayToBST(arr, 0, n - 1);
        System.out.println("Preorder traversal of constructed BST");
        tree.preOrder(root);
    }
}

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

Time Complexity: O(n)
Following is the recurrance 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.