DSA SERIES

- Learn Coding

Topic to be Covered today

Binary Search Tree

LETS START TODAY'S LECTURE

530. Minimum Absolute Difference in BST

```
class Solution {
public:
    void inorder(TreeNode* root, vector<int>& values) {
        if (root == NULL)
            return;
        inorder(root->left, values);
        values.push_back(root->val);
        inorder(root->right, values);
    }

    int getMinimumDifference(TreeNode* root) {

        vector<int> values;
        inorder(root, values);

        int minDiff = INT_MAX;

        for (int i = 1; i < values.size(); i++) {
            minDiff = min(minDiff, values[i] - values[i - 1]);
        }
        return minDiff;
            }
};
```

653. Two Sum IV - Input is a BST

```
class Solution {
public:
    void inorder(TreeNode* root, vector<int>& values) {
        if (root == NULL)
            return;
        inorder(root->left, values);
        values.push_back(root->val);
        inorder(root->right, values);
    }


    bool findTarget(TreeNode* root, int k) {
        vector<int> values;
        inorder(root, values);
        int n = values.size();


        int left = 0, right = n - 1;
```

```
while (left < right) {
            int sum = values[left] +
values[right];


            if (sum == k)
                return true;
            else if (sum < k)
                left++;
            else
                right--;
        }


        return false;
    }
};
```

700. Search in a Binary Search Tree

```
class Solution {
public:
    TreeNode* searchBST(TreeNode* root, int val) {
        if (root == NULL)
            return NULL;

        if (val == root->val)
            return root;

        else if (root->val < val)
            return searchBST(root->right, val);

        else
            return searchBST(root->left, val);

        return NULL;
    }
};
```

538. Convert BST to Greater Tree

```
class Solution {
public:
    int sum = 0;
    TreeNode* convertBST(TreeNode* root) {
        if (root == NULL)
            return NULL;


        convertBST(root->right);
        sum += root->val;
        root->val = sum;
        convertBST(root->left);
        return root;
    }
};
```

1305. All Elements in Two Binary Search Trees

```
class Solution {
public:
    vector<int> ans;

    void inorder(TreeNode* root) {
        if (root == NULL) {
            return;
        }

        inorder(root->left);
        ans.push_back(root->val);
        inorder(root->right);
    }
    vector<int> getAllElements(TreeNode* root1, TreeNode* root2) {
        inorder(root1);
        inorder(root2);
        sort(ans.begin(), ans.end());
        return ans;
    }
};
```

1382. Balance a Binary Search Tree

```
class Solution {
public:
    vector<int> values;


    void inorder(TreeNode* root) {
        if (root == NULL)
            return;
        inorder(root->left);
        values.push_back(root->val);
        inorder(root->right);
    }

    TreeNode* build(int left, int right) {
        if (left > right)
            return NULL;

        int mid = left + (right - left) / 2;


        TreeNode* root = new TreeNode(values[mid]);
```

```
root->left = build(left, mid - 1);
        root->right = build(mid + 1, right);
        return root;
    }


    TreeNode* balanceBST(TreeNode* root) {
        inorder(root);


        return build(0, values.size() - 1);
    }
};
```

Learn coding

THANK YOU