1. In-order traverse

```
class Solution
{
    static int isRepresentingBST(int arr[], int N)
    {

        // code here
        for(int i = 0; i + 1< N; i ++){
            if(arr[i] >= arr[i+1]){
                return 0;
            }
        }
        return 1;
    }
}
```

2. BST check

```
public class BST_Check
{
    //Function to check whether a Binary Tree is BST or not.
    boolean isBST(Node1 root)
    {
        // code here.
        int min = Integer.MIN_VALUE;
        int max = Integer.MAX_VALUE;
        return inOrder(root, min, max);

    }
    boolean inOrder (Node1 root, int min, int max){
        if(root == null) return true;
        if(root.data >= max || root.data <= min) return false;
        boolean right = inOrder(root.right, root.data, max);
        boolean left = inOrder(root.left, min, root.data);
        return (right && left);
    }
}
```

3. Median of BST

```
class Tree
{
    public static void inorder(Node root, List<Integer> list){
        if(root != null){
            inorder(root.left, list);
            list.add(root.data);
            inorder(root.right, list);
        }
    }
    public static float findMedian(Node root)
    {
        // code here.
        List<Integer> list = new ArrayList<>();
        inorder(root, list);
        int n = list.size();
        if(n % 2 == 0){
            return (float)(list.get(n/2) + list.get(n/2 - 1))/2;
        }else{
            return (float)list.get(n/2);
        }
    }

}
```

4. Count BST nodes lying in a given range

```
class Solution1
{
    //Function to count number of nodes in BST that lie in the given range.
    int getCount(Node root,int l, int h)
    {
        //Your code here
        List<Integer> list = new ArrayList<>();
        inorder(root, list);
        int n = list.size();
        int count = 0;
        for (int i = 0; i < n; i ++){
            if (list.get(i) >= l && list.get(i) <= h){
                count ++;
            }
        }
        return count;
    }
    public static void inorder(Node root, List<Integer> list){
        if(root != null){
            inorder(root.left, list);
            list.add(root.data);
            inorder(root.right, list);
        }
    }
}
```

5. BST: Kth smallest element

```
class Solution3 {
    // Return the Kth smallest element in the given BST
    public int KthSmallestElement(Node root, int K) {
        // Write your code here
        List<Integer> list = new ArrayList<>();
        inorder(root, list);
        int n = list.size();
        if (K > n){
            return - 1;
        }
        return list.get(K -1);
    }

    public static void inorder(Node root, List<Integer> list){
        if(root != null){
            inorder(root.left, list);
            list.add(root.data);
            inorder(root.right, list);
        }
    }
}
```

6. BST: Kth largest element

```
class Solution4
{
    // return the Kth the largest element in the given BST rooted at 'root'
    public int kthLargest(Node root,int K)
    {
        List<Integer> list = new ArrayList<>();
        inorder(root, list);
        int n = list.size();
        if (K > n){
            return - 1;
        }
        return list.get (n - K);
    }
```

```
    public static void inorder(Node root, List<Integer> list){
        if(root != null){
            inorder(root.left, list);
            list.add(root.data);
            inorder(root.right, list);
        }
    }
}
```