

Binary Search Tree Implementations:

1. CODE:

```
class Node{
    int k;
    Node l,r;
    public Node(int it){
        k=it;
        l=r=null;
    }
}

class bst_implementation_node{
    static Node insert(Node root, int k){
        if(root==null)
            return new Node(k);
        if(root.k==k)
            return root;
        if(k<root.k)
            root.l=insert(root.l,k);
        else
            root.r=insert(root.r,k);
        return root;
    }
    static void inorder(Node root){
        if(root!=null){
            inorder(root.l);
            System.out.print(root.k+" ");
            inorder(root.r);
        }
    }
    public static void main(String[] args){
        Node root=null;
        root=insert(root, 50);
        root=insert(root, 30);
        root=insert(root, 20);
        root=insert(root, 40);
        root=insert(root, 70);
        root=insert(root, 60);
        root=insert(root, 80);
        inorder(root);
    }
}
```

OUTPUT:

```
[Running] cd "c:\Data\Knowledge\DSA\Programs\Bst\" && javac bst_implementation_node.java && java
bst_implementation_node
20 30 40 50 60 70 80
```

Time complexity: $O(n)$

2. CODE:

```
import java.util.TreeSet;
class bst_implementation_treeset{
    public static void main(String[] args){
        TreeSet<Integer> bst=new TreeSet<>();
        bst.add(50);
        bst.add(30);
        bst.add(20);
        bst.add(70);
        bst.add(40);
        bst.add(80);
        bst.add(60);
        System.out.println(bst);
    }
}
```

OUTPUT:

```
[Running] cd "c:\Data\Knowledge\DSA\Programs\Bst\" && javac bst_implementation_treeset.java && java
bst_implementation_treeset
[20, 30, 40, 50, 60, 70, 80]
```

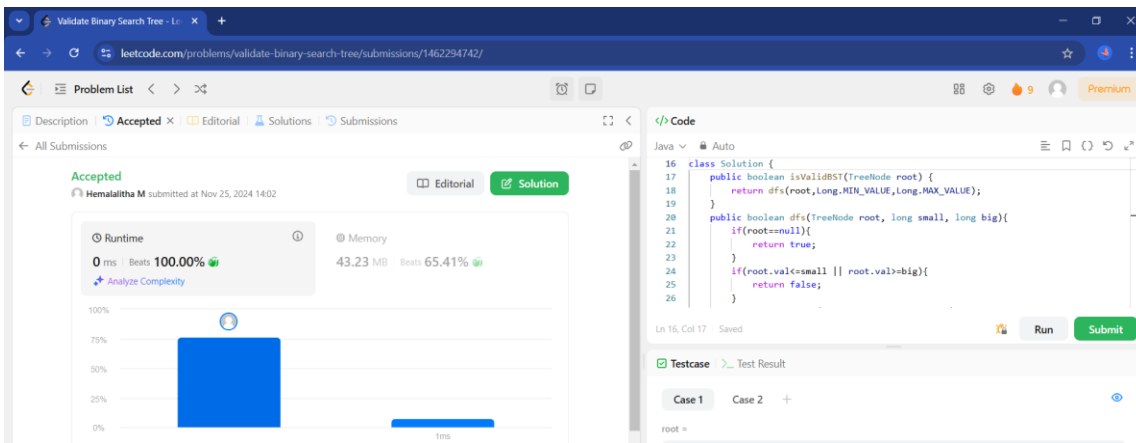
Time complexity: $O(\log n)$

3. Validate binary search tree:

CODE:

```
class Solution {
    public boolean isValidBST(TreeNode root) {
        return dfs(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }
    public boolean dfs(TreeNode root, long small, long big){
        if(root==null){
            return true;
        }
        if(root.val<=small || root.val>=big){
            return false;
        }
        boolean left=dfs(root.left,small,root.val);
        boolean right=dfs(root.right,root.val,big);
        if(left && right){
            return true;
        }
        return false;
    }
}
```

OUTPUT:



Time complexity: $O(n)$

4. Left view of BST:

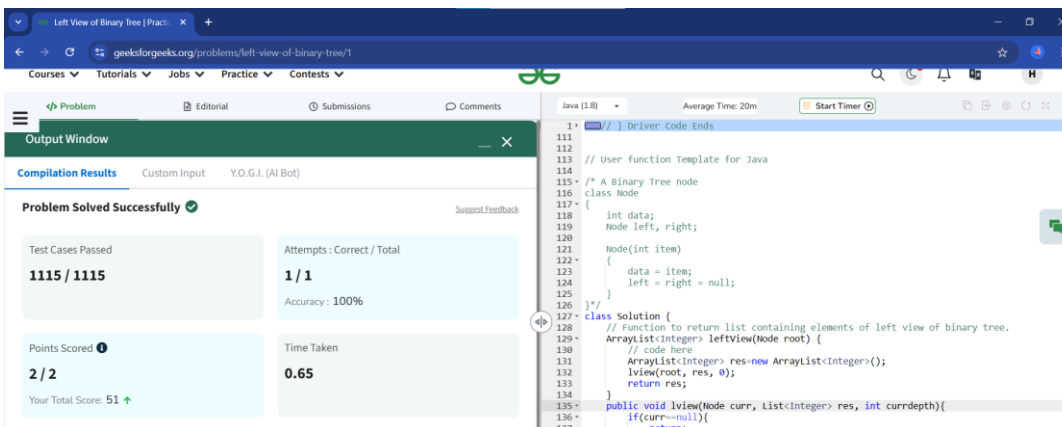
CODE:

```

class Solution {
    ArrayList<Integer> leftView(Node root) {
        ArrayList<Integer> res=new ArrayList<Integer>();
        lview(root, res, 0);
        return res;
    }
    public void lview(Node curr, List<Integer> res, int currdepth){
        if(curr==null){
            return;
        }
        if(currdepth==res.size()){
            res.add(curr.data);
        }
        lview(curr.left, res, currdepth+1);
        lview(curr.right, res, currdepth+1);
    }
}

```

OUTPUT:



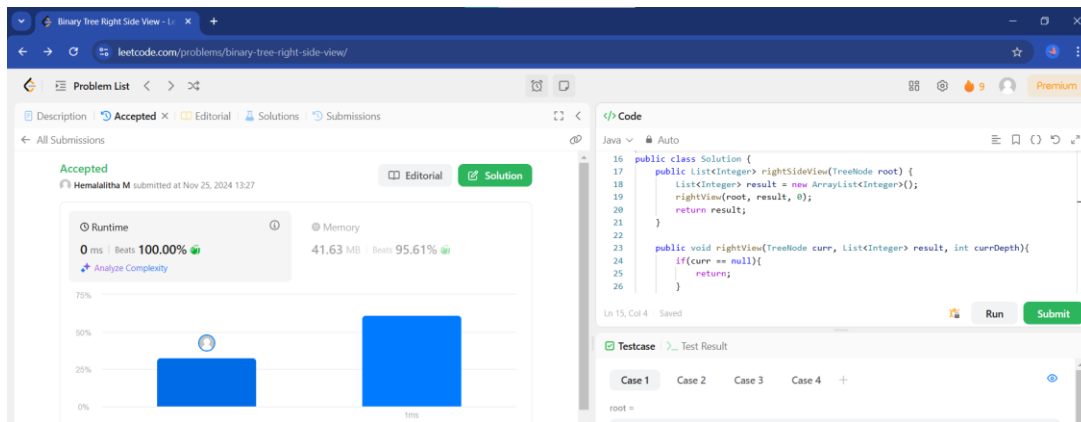
Time complexity: $O(n)$

5. Right view of BST:

CODE:

```
public class Solution {
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> result = new ArrayList<Integer>();
        rightView(root, result, 0);
        return result;
    }
    public void rightView(TreeNode curr, List<Integer> result, int currDepth){
        if(curr == null){
            return;
        }
        if(currDepth == result.size()){
            result.add(curr.val);
        }
        rightView(curr.right, result, currDepth + 1);
        rightView(curr.left, result, currDepth + 1);
    }
}
```

OUTPUT:



Time complexity: $O(n)$

6. Top view of BST:

CODE:

```
class Pair<k,v>{
    k key;
    v value;
    Pair(k key, v value){
        this.key=key;
        this.value=value;
    }
    k getKey(){
        return key;
    }
    v getValue(){
        return value;
    }
}

class Solution {
```

```

static ArrayList<Integer> topView(Node root) {
    ArrayList<Integer> a=new ArrayList<>();
    if(root==null) return a;
    Map<Integer, Integer> m=new TreeMap<>();
    Queue<Pair<Node,Integer>> q=new LinkedList<>();
    q.add(new Pair<>(root,0));
    while(!q.isEmpty()){
        Pair<Node, Integer> p=q.poll();
        Node node=p.getKey();
        int l=p.getValue();
        if(!m.containsKey(l)){
            m.put(l,node.data);
        }
        if(node.left!=null){
            q.add(new Pair<>(node.left,l-1));
        }
        if(node.right!=null){
            q.add(new Pair<>(node.right, l+1));
        }
    }
    for(int v:m.values()){
        a.add(v);
    }
    return a;
}
}

```

OUTPUT:

The screenshot shows a web IDE interface with the following details:

- Problem:** Top View of Binary Tree | Practice
- Compilation Results:** Problem Solved Successfully
- Test Cases Passed:** 1111 / 1111
- Attempts:** Correct / Total: 1 / 1
- Accuracy:** 100%
- Points Scored:** 4 / 4
- Time Taken:** 0.68
- Your Total Score:** 57
- Code Editor:** Shows the Java solution for the problem, including the `topView` function and its implementation.

Time complexity: $O(n \log n)$

7. Bottom view of BST:

CODE:

```

class Pair{
    Node node;
    int hd;
    Pair(Node node, int hd){
        this.node=node;
        this.hd=hd;
    }
}

```

```

}
class Solution
{
    public ArrayList <Integer> bottomView(Node root)
    {
        // Code here
        if(root==null) return new ArrayList<>();
        Map<Integer, Integer> hm=new TreeMap<>();
        Queue<Pair> q=new LinkedList<>();
        q.add(new Pair(root,0));
        while(!q.isEmpty()){
            Node curr=q.peek().node;
            int hd=q.peek().hd;
            q.poll();
            hm.put(hd, curr.data);
            if(curr.left!=null){
                q.add(new Pair(curr.left, hd-1));
            }
            if(curr.right!=null){
                q.add(new Pair(curr.right, hd+1));
            }
        }
        ArrayList<Integer> res=new ArrayList<>();
        for(int val:hm.values()){
            res.add(val);
        }
        return res;
    }
}

```

OUTPUT:

The screenshot shows a coding platform interface. On the left, a sidebar indicates the problem is solved successfully with 1115/1115 test cases passed, 100% accuracy, 4/4 points scored, and a time taken of 1.25. The main editor displays the Java code for the 'Bottom View of Binary Tree' problem. The code uses a TreeMap to store node data by horizontal distance (hd) and a queue for BFS traversal.

Time complexity: $O(n \log n)$

8. Segment tree:

CODE:

```

class GfG
{
    static int st[];

```

```

public static int[] constructST(int arr[], int n)
{
    st = new int[4 * n];
    buildST(arr, 0, n - 1, 0);
    return st;
}

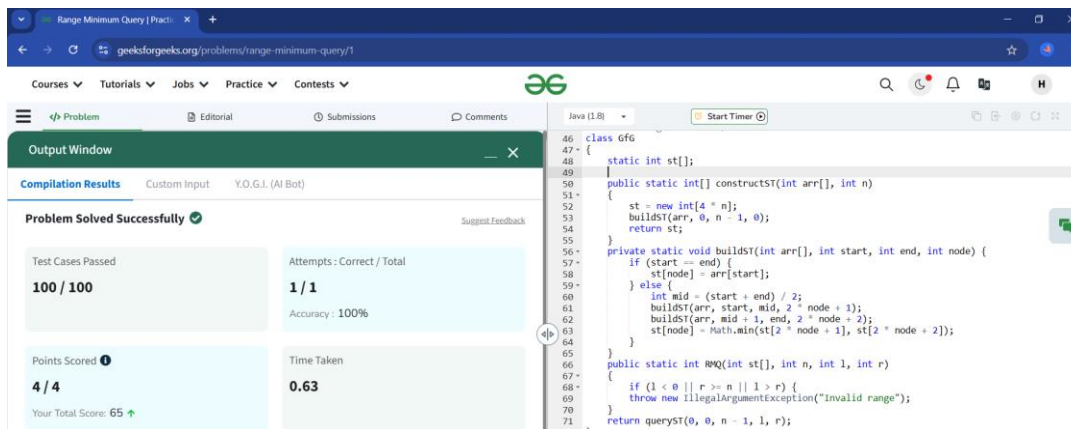
private static void buildST(int arr[], int start, int end, int node) {
    if (start == end) {
        st[node] = arr[start];
    } else {
        int mid = (start + end) / 2;
        buildST(arr, start, mid, 2 * node + 1);
        buildST(arr, mid + 1, end, 2 * node + 2);
        st[node] = Math.min(st[2 * node + 1], st[2 * node + 2]);
    }
}

public static int RMQ(int st[], int n, int l, int r)
{
    if (l < 0 || r >= n || l > r) {
        throw new IllegalArgumentException("Invalid range");
    }
    return queryST(0, 0, n - 1, l, r);
}

private static int queryST(int node, int start, int end, int l, int r) {
    if (start > r || end < l) {
        return Integer.MAX_VALUE;
    }
    if (start >= l && end <= r) {
        return st[node];
    }
    int mid = (start + end) / 2;
    int leftMin = queryST(2 * node + 1, start, mid, l, r);
    int rightMin = queryST(2 * node + 2, mid + 1, end, l, r);
    return Math.min(leftMin, rightMin);
}
}

```

OUTPUT:



The screenshot shows a web IDE interface for solving the Range Minimum Query problem. The left sidebar contains the 'Output Window' with the following details:

- Compilation Results:** Problem Solved Successfully
- Test Cases Passed:** 100 / 100
- Attempts:** Correct / Total: 1 / 1
- Accuracy:** 100%
- Points Scored:** 4 / 4
- Your Total Score:** 65
- Time Taken:** 0.63

The right sidebar displays the Java code for the solution, which is a segment of the code provided in the first block. The code defines a class 'GfG' with methods 'constructST', 'buildST', 'RMQ', and 'queryST'.

Time complexity: $O(n)$