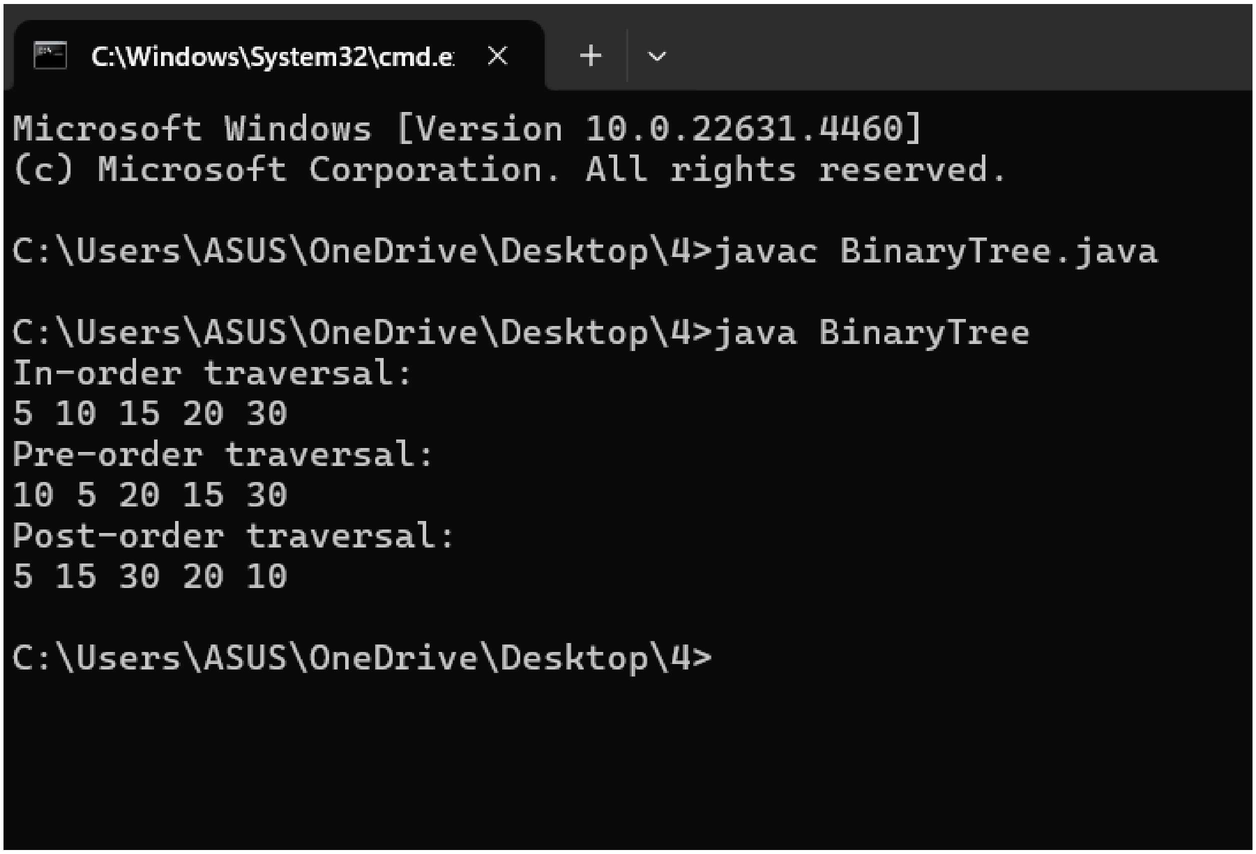
1)Binary tree

```
public class BinaryTree {
    static class TreeNode {
        int val;
        TreeNode left, right;
        TreeNode(int x) {
            val = x;
        }
    }

    TreeNode root;

    public BinaryTree() {
        root = null;
    }

    public void insert(int val) {
        root = insertRecursive(root, val);
    }

    private TreeNode insertRecursive(TreeNode node, int val) {
        if (node == null) {
            return new TreeNode(val);
        }
        if (val < node.val) {
            node.left = insertRecursive(node.left, val);
        } else if (val > node.val) {
            node.right = insertRecursive(node.right, val);
        }
        return node;
    }

    public void inorder() {
        inorderRecursive(root);
```

```
}

private void inorderRecursive(TreeNode node) {
    if (node != null) {
        inorderRecursive(node.left);
        System.out.print(node.val + " ");
        inorderRecursive(node.right);
    }
}

public void preorder() {
    preorderRecursive(root);
}

private void preorderRecursive(TreeNode node) {
    if (node != null) {
        System.out.print(node.val + " ");
        preorderRecursive(node.left);
        preorderRecursive(node.right);
    }
}

public void postorder() {
    postorderRecursive(root);
}

private void postorderRecursive(TreeNode node) {
    if (node != null) {
        postorderRecursive(node.left);
        postorderRecursive(node.right);
        System.out.print(node.val + " ");
    }
}

public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();
        tree.insert(10);
        tree.insert(20);
        tree.insert(5);
        tree.insert(15);
        tree.insert(30);

        System.out.println("In-order traversal:");
        tree.inorder();          System.out.println();

        System.out.println("Pre-order traversal:");
        tree.preorder();
        System.out.println();

        System.out.println("Post-order traversal:");
        tree.postorder();
        System.out.println();
    }
}
```

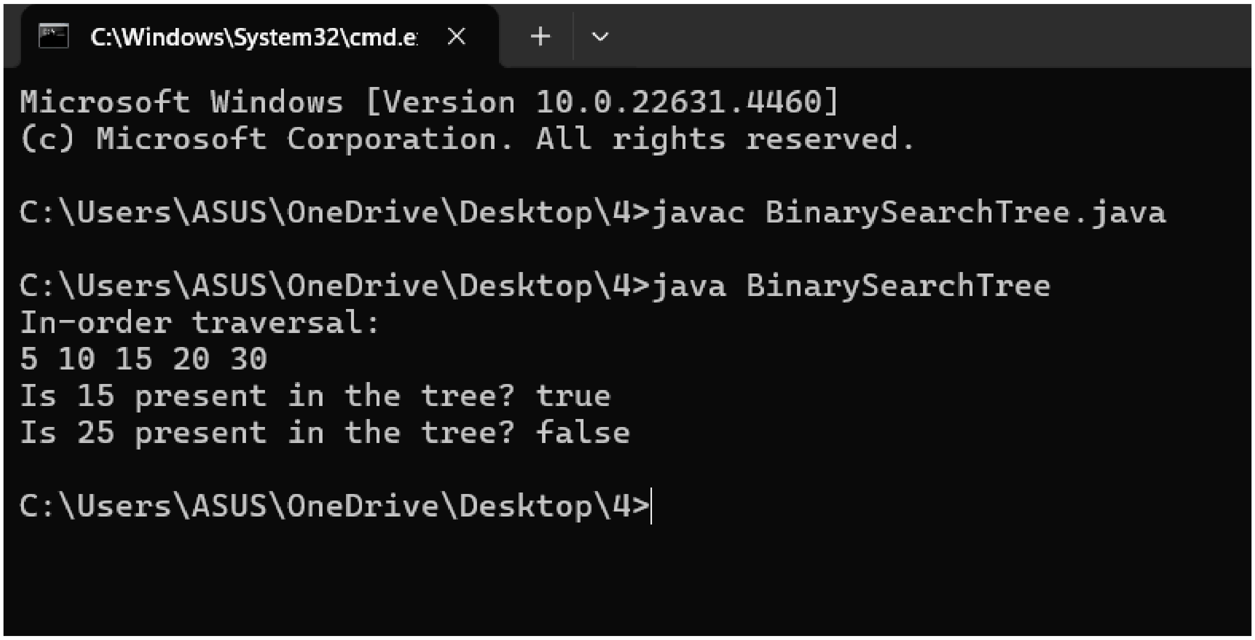


```
Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS\OneDrive\Desktop\4>javac BinaryTree.java

C:\Users\ASUS\OneDrive\Desktop\4>java BinaryTree
In-order traversal:
5 10 15 20 30
Pre-order traversal:
10 5 20 15 30
Post-order traversal:
5 15 30 20 10

C:\Users\ASUS\OneDrive\Desktop\4>
```

Time:O(n)-traversal
insertion-O(logn)

2)BInary search tree

```
public class BinarySearchTree {
    static class TreeNode {
        int val;
        TreeNode left, right;
        TreeNode(int x) {
            val = x;
        }
    }

    private TreeNode root;

    public BinarySearchTree() {
        root = null;
    }

    public void insert(int val) {
        root = insertRecursive(root, val);
    }

    private TreeNode insertRecursive(TreeNode node, int val) {
        if (node == null) {
            return new TreeNode(val);
        }
        if (val < node.val) {
            node.left = insertRecursive(node.left, val);
        } else if (val > node.val) {
            node.right = insertRecursive(node.right, val);
        }
        return node;
    }

    public boolean search(int val) {
        return searchRecursive(root, val);
    }

    private boolean searchRecursive(TreeNode node, int val) {
        if (node == null) {
            return false;
        }
        if (val == node.val) {
            return true;
        }
        return val < node.val ? searchRecursive(node.left, val) :
searchRecursive(node.right, val);
    }

    public void inorder() {
        inorderRecursive(root);
    }

    private void inorderRecursive(TreeNode node) {
        if (node != null) {
            inorderRecursive(node.left);
            System.out.print(node.val + " ");
            inorderRecursive(node.right);
        }
    }

    public static void main(String[] args) {
        BinarySearchTree bst = new BinarySearchTree();
        bst.insert(10);
        bst.insert(20);
        bst.insert(5);
        bst.insert(15);
        bst.insert(30);

        System.out.println("In-order traversal:");
        bst.inorder();
        System.out.println();

        int target = 15;
        System.out.println("Is " + target + " present in the tree? " +
bst.search(target));
        target = 25;
        System.out.println("Is " + target + " present in the tree? " +
bst.search(target));     }
}
```



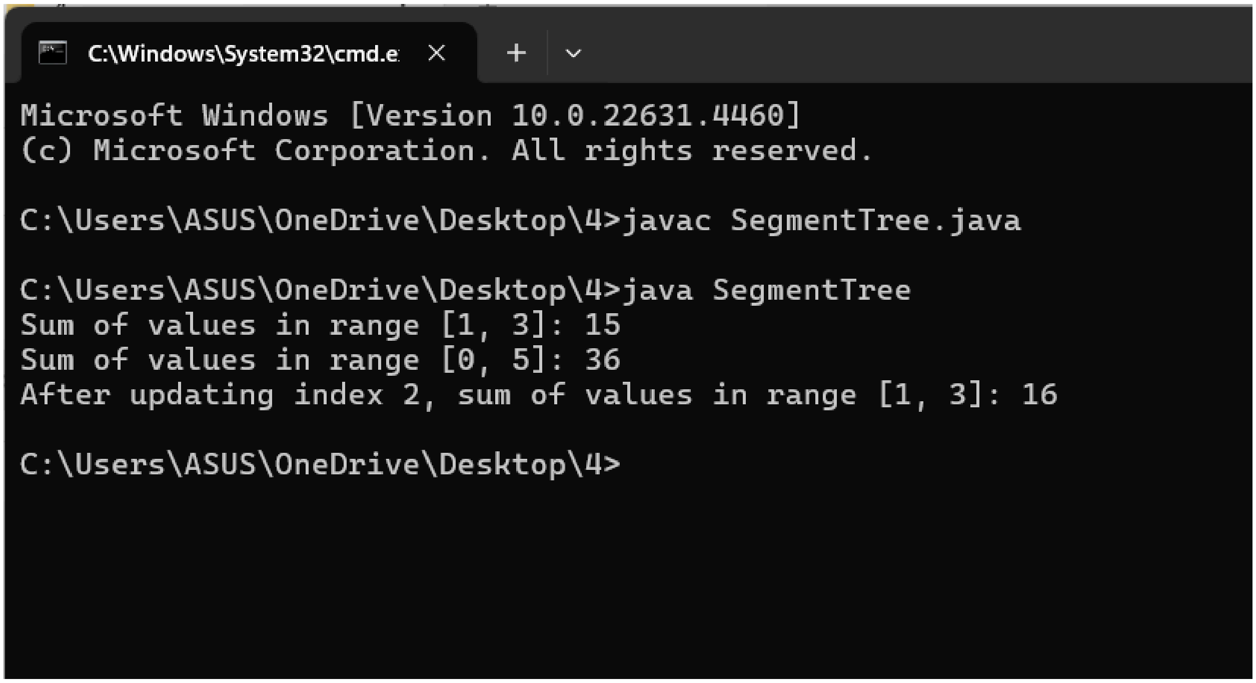
```
Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS\OneDrive\Desktop\4>javac BinarySearchTree.java

C:\Users\ASUS\OneDrive\Desktop\4>java BinarySearchTree
In-order traversal:
5 10 15 20 30
Is 15 present in the tree? true
Is 25 present in the tree? false

C:\Users\ASUS\OneDrive\Desktop\4>
```

Time:O(n)-inorder traversal
Insertion and Search: O(log n)

3)segment tree

```
public class SegmentTree {
    private int[] tree;
    private int n;

    public SegmentTree(int[] arr) {
        n = arr.length;
        tree = new int[4 * n];
        build(arr, 0, 0, n - 1);
    }

    private void build(int[] arr, int node, int start, int end) {
        if (start == end) {
            tree[node] = arr[start];
        } else {
            int mid = start + (end - start) / 2;
            build(arr, 2 * node + 1, start, mid);
            build(arr, 2 * node + 2, mid + 1, end);
            tree[node] = tree[2 * node + 1] + tree[2 * node + 2];
        }
    }

    public void update(int idx, int val) {
        update(0, 0, n - 1, idx, val);
    }

    private void update(int node, int start, int end, int idx, int val) {
        if (start == end) {
            tree[node] = val;
        } else {
            int mid = start + (end - start) / 2;
            if (idx <= mid) {
                update(2 * node + 1, start, mid, idx, val);
            } else {
                update(2 * node + 2, mid + 1, end, idx, val);
            }
            tree[node] = tree[2 * node + 1] + tree[2 * node + 2];
        }
    }

    public int query(int left, int right) {
        return query(0, 0, n - 1, left, right);
    }

    private int query(int node, int start, int end, int left, int right) {
        if (right < start || end < left) {
            return 0;         }
        if (left <= start && end <= right) {
            return tree[node];
        }
             int mid = start + (end - start) / 2;
        int leftSum = query(2 * node + 1, start, mid, left, right);
```

```
        int rightSum = query(2 * node + 2, mid + 1, end, left, right);
        return leftSum + rightSum;
    }

    public static void main(String[] args) {
        int[] arr = {1, 3, 5, 7, 9, 11};
        SegmentTree segmentTree = new SegmentTree(arr);

        System.out.println("Sum of values in range [1, 3]: " +
segmentTree.query(1, 3));         System.out.println("Sum of values in range
[0, 5]: " + segmentTree.query(0, 5));
        segmentTree.update(2, 6);
        System.out.println("After updating index 2, sum of values in range [1,
3]: " + segmentTree.query(1, 3));
}
}
```

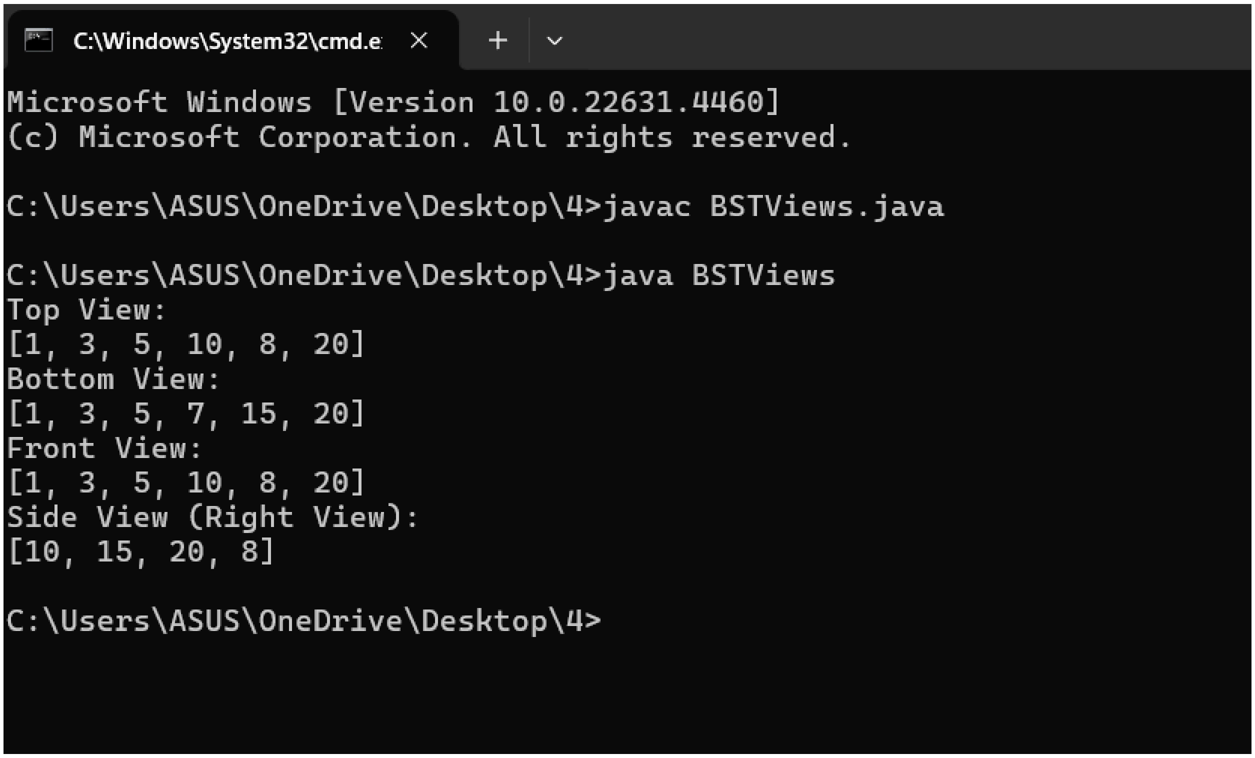


```
Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS\OneDrive\Desktop\4>javac SegmentTree.java

C:\Users\ASUS\OneDrive\Desktop\4>java SegmentTree
Sum of values in range [1, 3]: 15
Sum of values in range [0, 5]: 36
After updating index 2, sum of values in range [1, 3]: 16

C:\Users\ASUS\OneDrive\Desktop\4>
```

Time:Build: O(n)
Update: O(log n)
Query: O(log n)

4)front,top,side,bottom views of bst tree

```
import java.util.*;

public class BSTViews {

    static class TreeNode {
        int val;
        TreeNode left, right;
        TreeNode(int x) {
            val = x;
        }
    }

    private static Map<Integer, Integer> topViewMap = new TreeMap<>();
    private static Map<Integer, Integer> bottomViewMap = new
TreeMap<>();
    private static Map<Integer, Integer> frontViewMap = new TreeMap<>();
    private static List<Integer> sideView = new ArrayList<>();

    public static void main(String[] args) {
        TreeNode root = new TreeNode(10);
        root.left = new TreeNode(5);
        root.right = new TreeNode(15);
        root.left.left = new TreeNode(3);
        root.left.right = new TreeNode(7);
        root.right.right = new TreeNode(20);
        root.left.left.left = new TreeNode(1);
        root.left.right.right = new TreeNode(8);

        System.out.println("Top View:");
        topView(root);
        System.out.println(topViewMap.values());
        System.out.println("Bottom View:");
        bottomView(root);
        System.out.println(bottomViewMap.values());

        System.out.println("Front View:");
        frontView(root, 0, 0);
        System.out.println(frontViewMap.values());

        System.out.println("Side View (Right View):");
        sideView(root, 0);
        System.out.println(sideView);
    }


    public static void topView(TreeNode root) {
        topViewMap.clear();
        topViewHelper(root, 0, 0);
    }

    private static void topViewHelper(TreeNode node, int horizontalDistance,
int level) {
        if (node == null) return;

        if (!topViewMap.containsKey(horizontalDistance)) {
            topViewMap.put(horizontalDistance, node.val);
        }
        topViewHelper(node.left, horizontalDistance - 1, level + 1);
        topViewHelper(node.right, horizontalDistance + 1, level + 1);
    }


    public static void bottomView(TreeNode root) {
        bottomViewMap.clear();
        bottomViewHelper(root, 0, 0);
    }
    private static void bottomViewHelper(TreeNode node, int horizontalDistance, int level) {
        if (node == null) return;

        bottomViewMap.put(horizontalDistance, node.val);
        bottomViewHelper(node.left, horizontalDistance - 1, level + 1);
        bottomViewHelper(node.right, horizontalDistance + 1, level + 1);
    }


    public static void frontView(TreeNode root, int horizontalDistance, int level) {
        if (root == null) return;

        frontViewMap.putIfAbsent(horizontalDistance, root.val);
        frontView(root.left, horizontalDistance - 1, level + 1);
        frontView(root.right, horizontalDistance + 1, level + 1);
    }


    public static void sideView(TreeNode root, int level) {
        if (root == null) return;

        if (level == sideView.size()) {
            sideView.add(root.val);
        }

        sideView(root.right, level + 1);
        sideView(root.left, level + 1);
    }
}
```



```
Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS\OneDrive\Desktop\4>javac BSTViews.java

C:\Users\ASUS\OneDrive\Desktop\4>java BSTViews
Top View:
[1, 3, 5, 10, 8, 20]
Bottom View:
[1, 3, 5, 7, 15, 20]
Front View:
[1, 3, 5, 10, 8, 20]
Side View (Right View):
[10, 15, 20, 8]

C:\Users\ASUS\OneDrive\Desktop\4>
```

time:O(n).