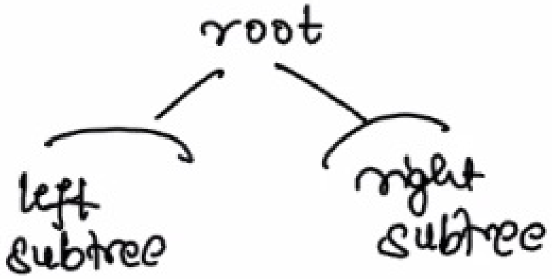
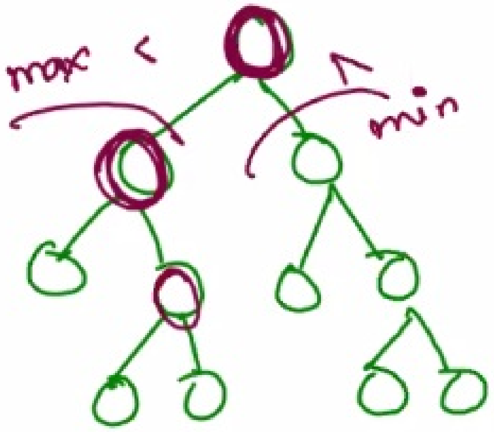


for BST

→ # all nodes in left subtree is smaller tham root.data

all nodes in right subtree is greater than root.data

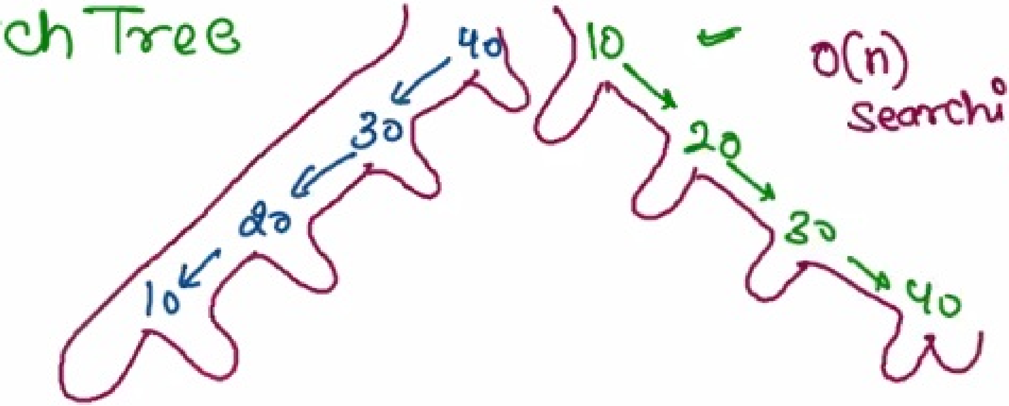
→ these are valid for all node in BST.



Benefits. → * searching.] Based on searching.
* store
* value based problem.

BST → Binary Search Tree

AVL → Balanced BST



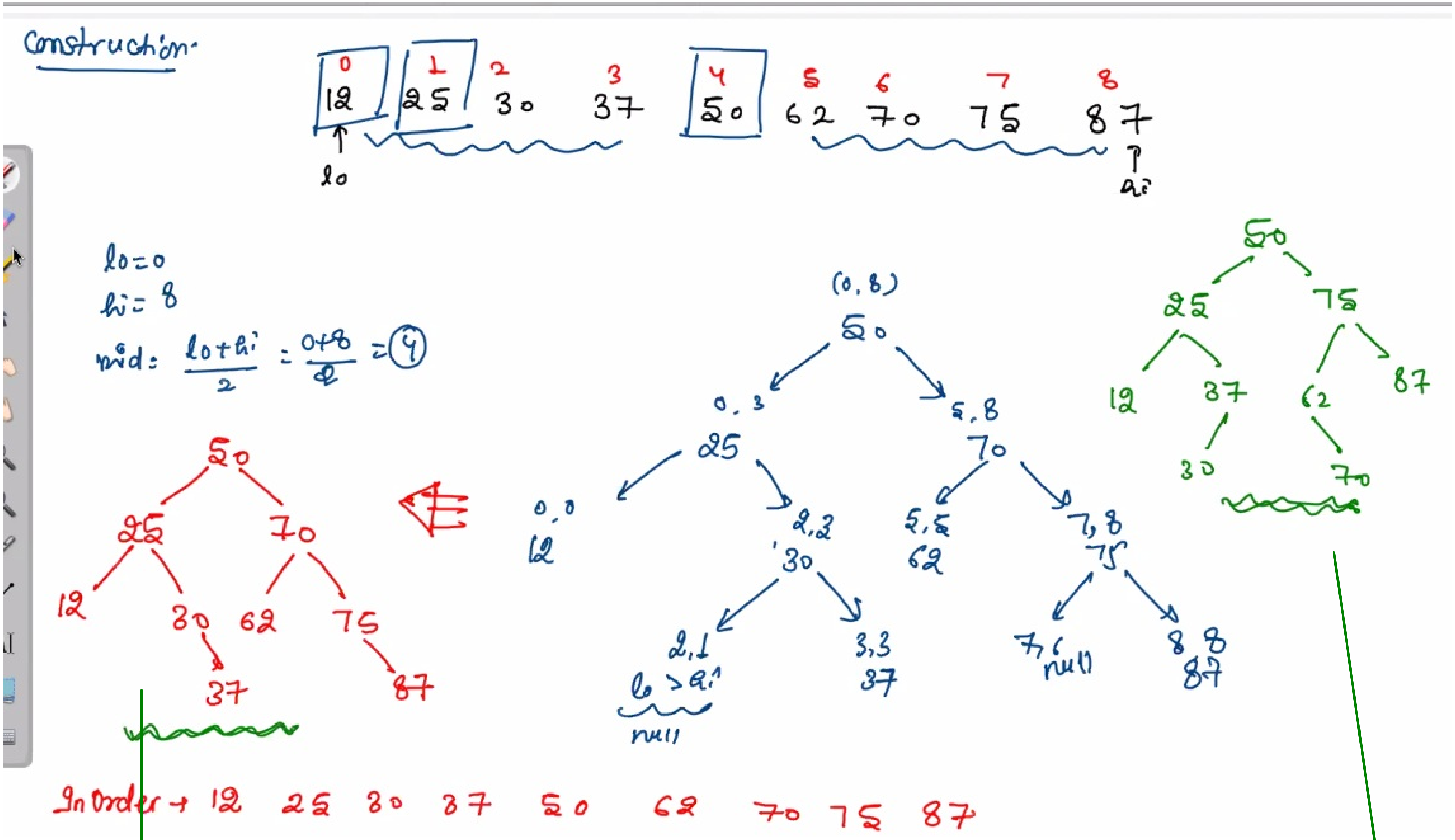
O(n) searching

Construction.

Inorder → Sorted

10 20 30 40

10, 20, 30, 40



Construction:

0	1	2	3	4	5	6	7	8
12	25	30	37	50	62	70	75	87

lo = 0

hi = 8

$\text{mid} = \frac{lo+hi}{2} = \frac{0+8}{2} = \boxed{4}$

Inorder → 12 25 30 37 50 62 70 75 87

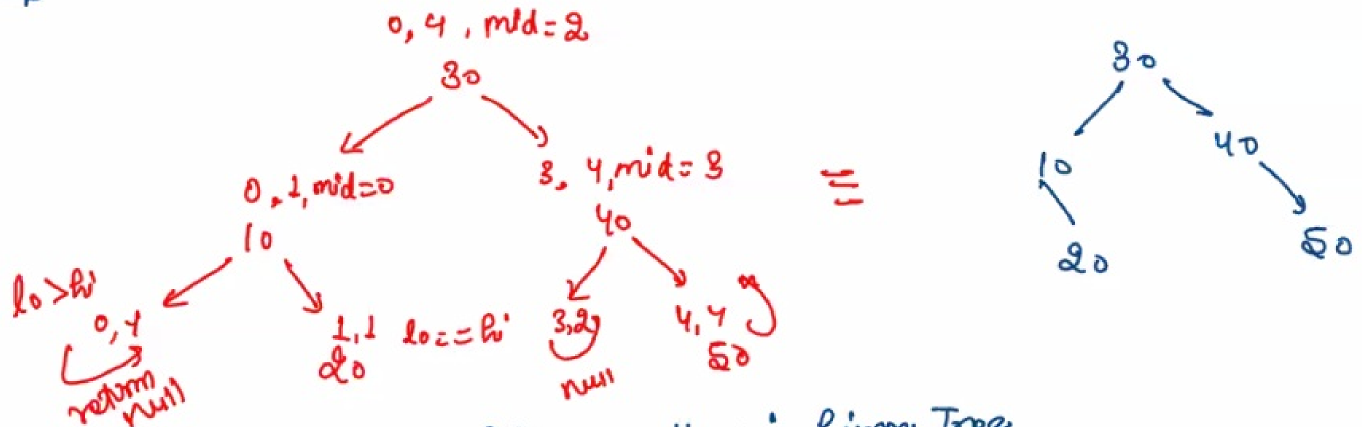
indorder

indorder

tree are diffrent but inroder wil be same. so that is if inorder is given , you can have multiple BST

	0	1	2	3	4
data →	10	20	30	40	50

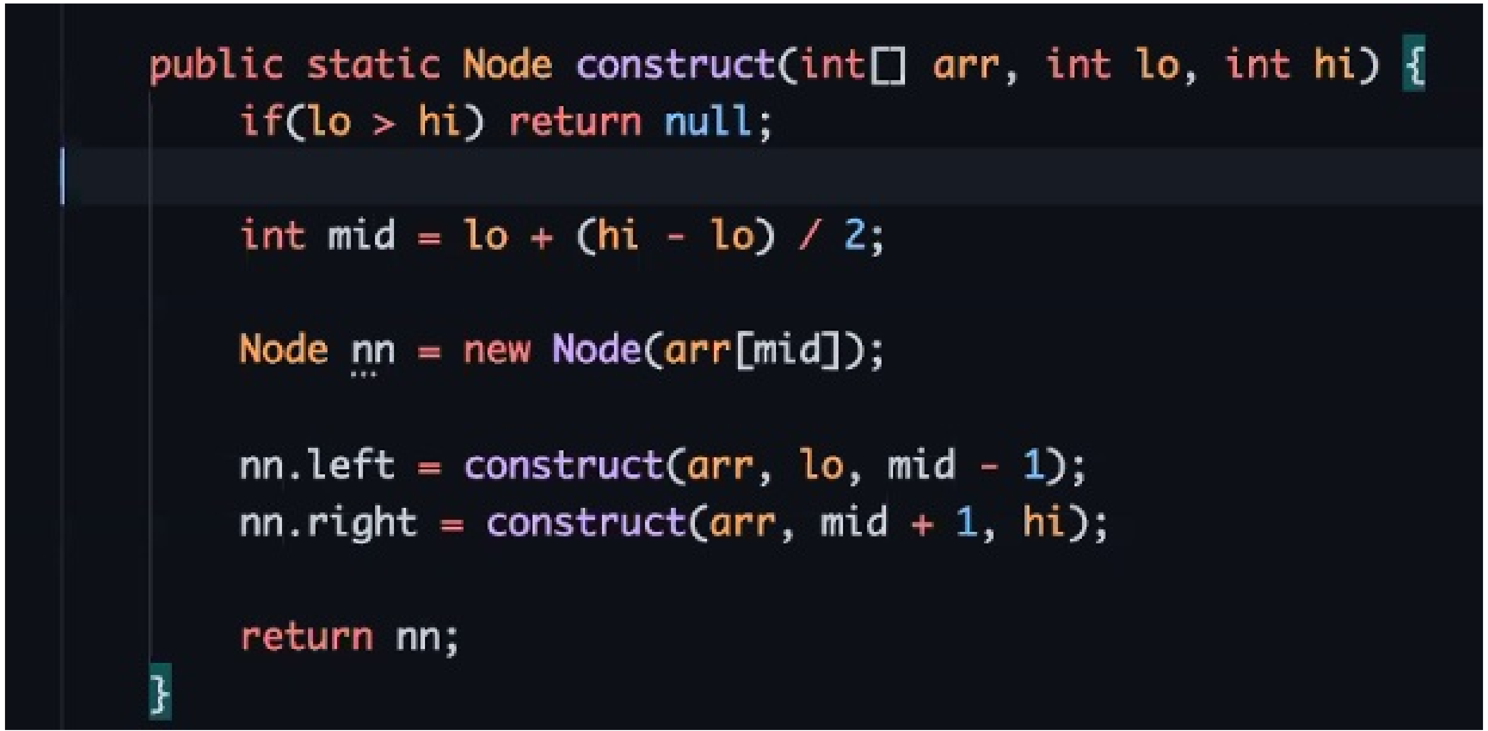
construction →



Same In BST as well as in Binary Tree

Structured based → → size
→ height
→ Diameter

Value based problem → min, max, find



```
public static Node construct(int[] arr, int lo, int hi) {
    if(lo > hi) return null;

    int mid = lo + (hi - lo) / 2;

    Node nn = new Node(arr[mid]);

    nn.left = construct(arr, lo, mid - 1);
    nn.right = construct(arr, mid + 1, hi);

    return nn;
}
```

max in BST → Right most Node

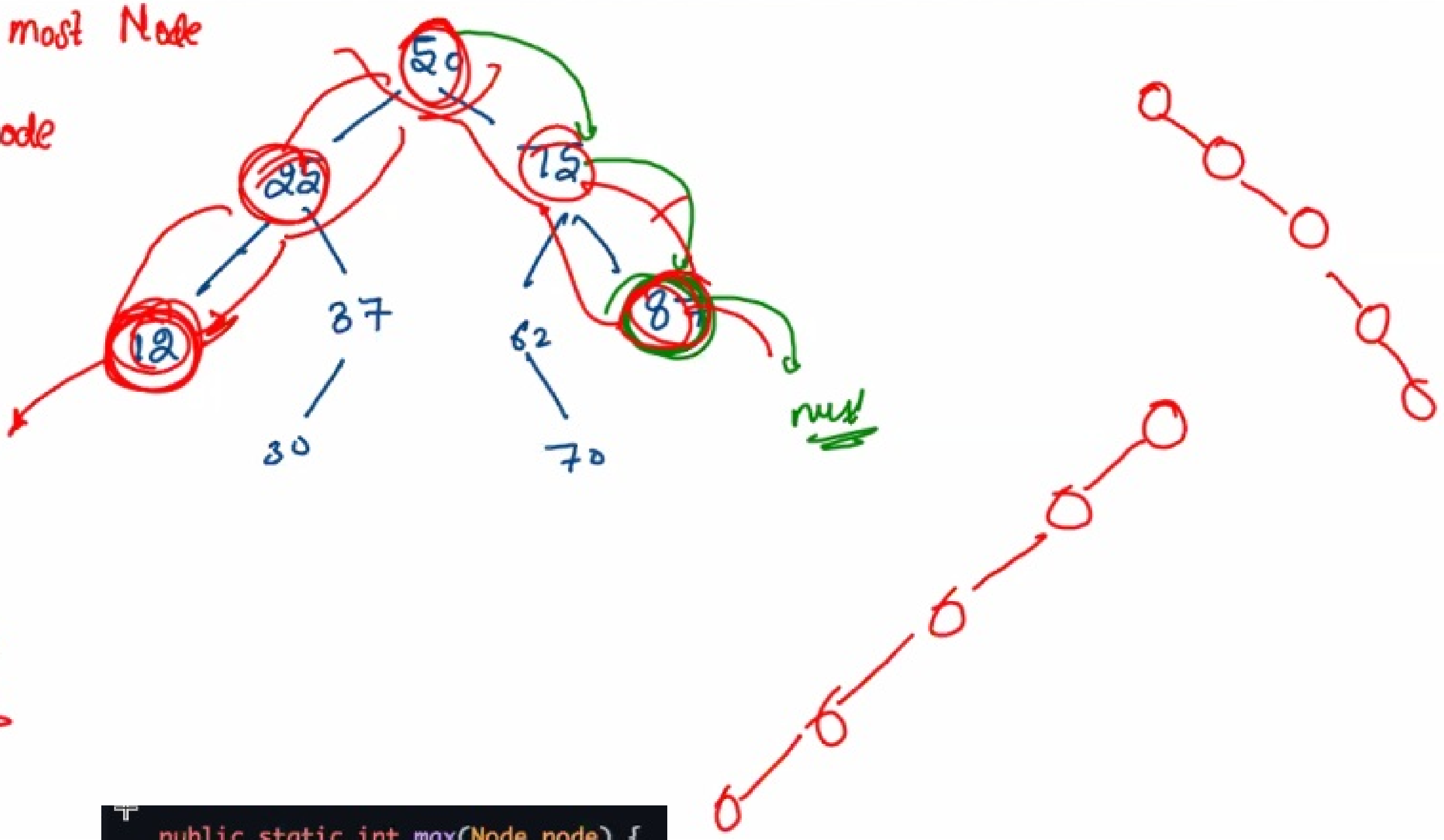
min → left most Node

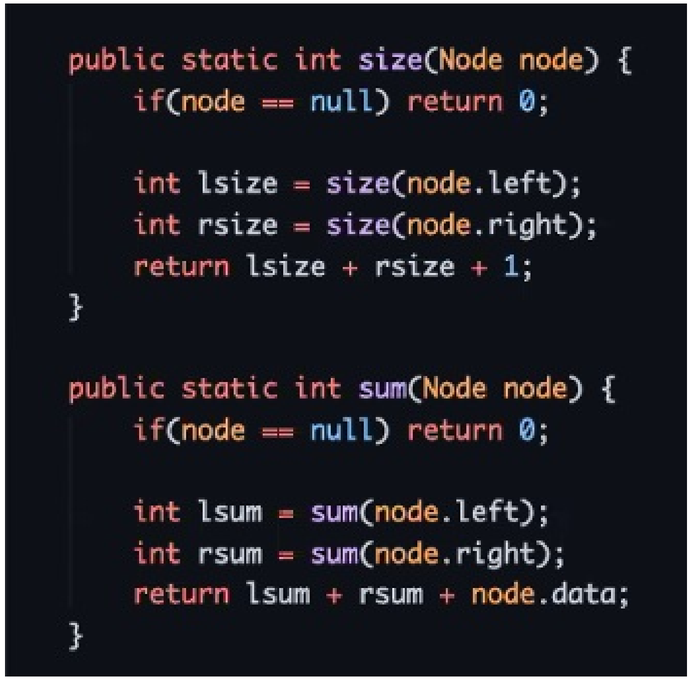
O(h)

h is height

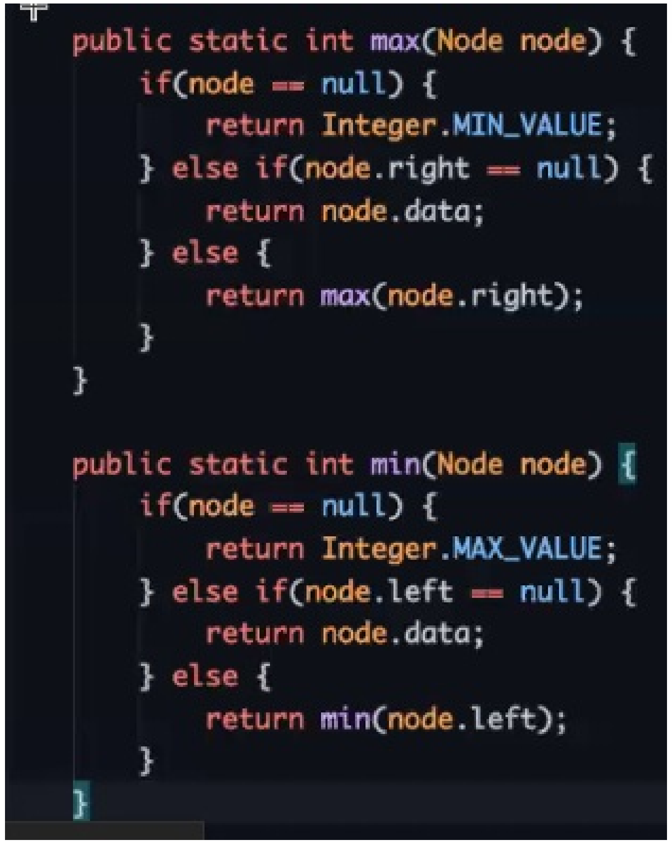
AVL → logn.

height = logn.

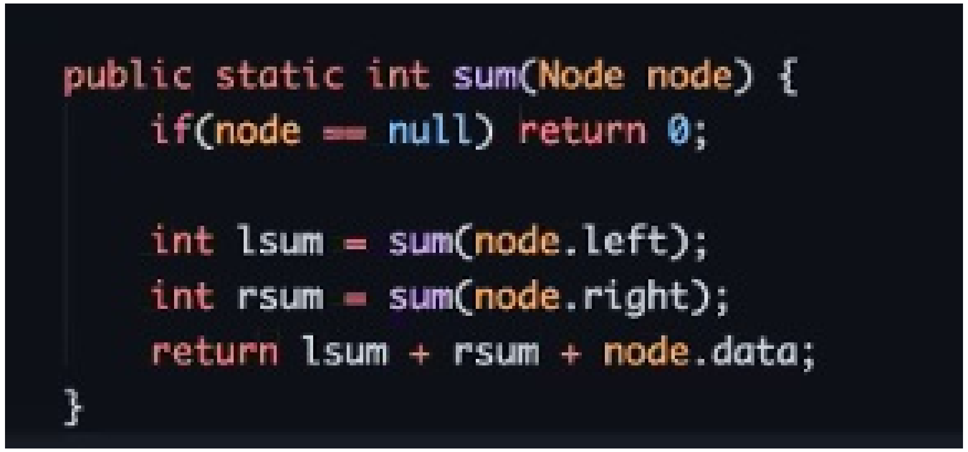




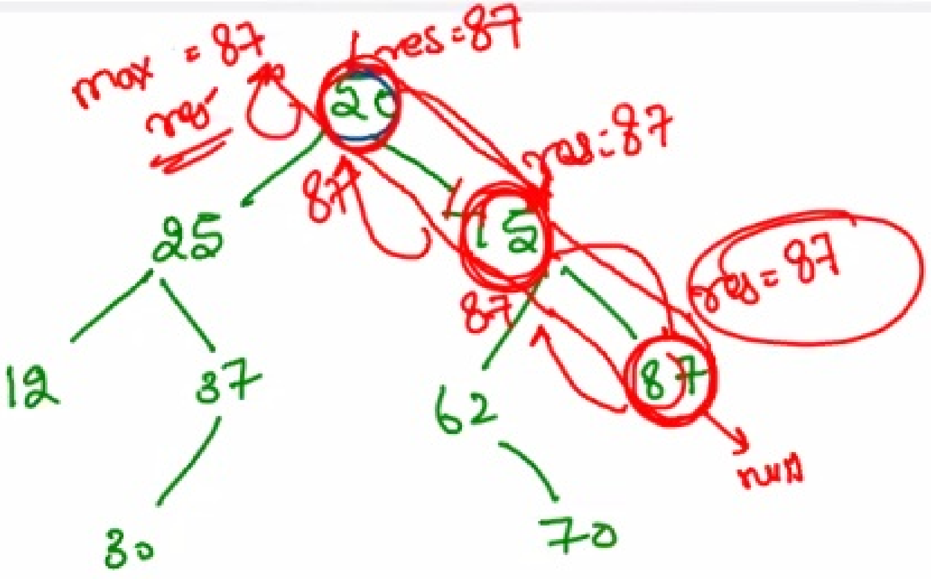
```java
public static int size(Node node) {
    if(node == null) return 0;

    int lsize = size(node.left);
    int rsize = size(node.right);
    return lsize + rsize + 1;
}

public static int sum(Node node) {
    if(node == null) return 0;

    int lsum = sum(node.left);
    int rsum = sum(node.right);
    return lsum + rsum + node.data;
}
```

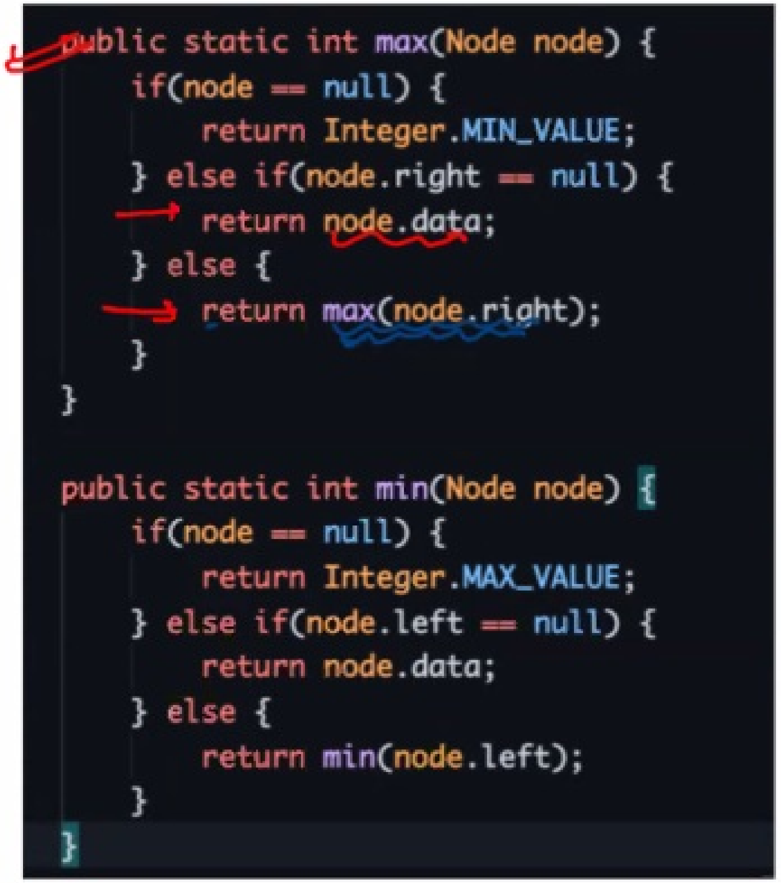


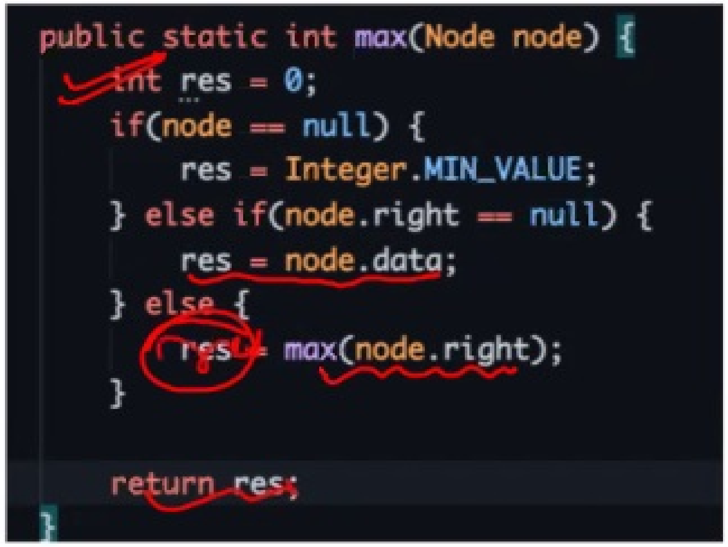
```java
public static int max(Node node) {
    if(node == null) {
        return Integer.MIN_VALUE;
    } else if(node.right == null) {
        return node.data;
    } else {
        return max(node.right);
    }
}

public static int min(Node node) {
    if(node == null) {
        return Integer.MAX_VALUE;
    } else if(node.left == null) {
        return node.data;
    } else {
        return min(node.left);
    }
}
```



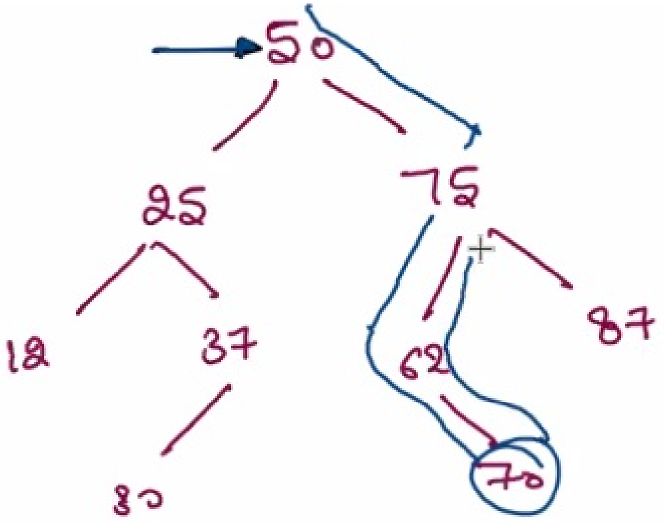
```java
public static int sum(Node node) {
    if(node == null) return 0;

    int lsum = sum(node.left);
    int rsum = sum(node.right);
    return lsum + rsum + node.data;
}
```




```java
public static int max(Node node) {
    if(node == null) {
        return Integer.MIN_VALUE;
    } else if(node.right == null) {
        return node.data;
    } else {
        return max(node.right);
    }
}

public static int min(Node node) {
    if(node == null) {
        return Integer.MAX_VALUE;
    } else if(node.left == null) {
        return node.data;
    } else {
        return min(node.left);
    }
}
```


```java
public static int max(Node node) {
    int res = 0;
    if(node == null) {
        res = Integer.MIN_VALUE;
    } else if(node.right == null) {
        res = node.data;
    } else {
        res = max(node.right);
    }

    return res;
}
```

find

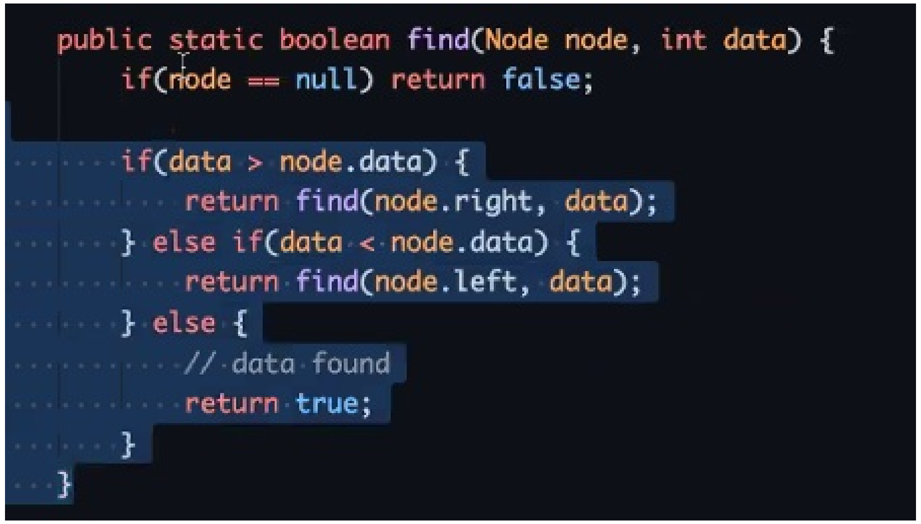


dtf = 70

dtf = 35.

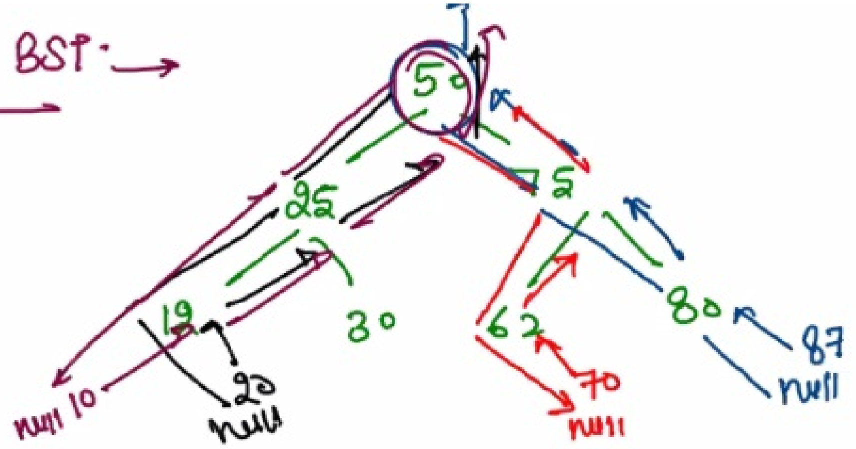
```
if( data > root.data) {
    // Right side.
    return  find(root.right, dtf);
} else if (data < root.data) {
    // left side
    return  find(root.left, dtf);
} else { // data ==  root.data
    //data found
    return  true;
}
```

[redacted]



```
public static boolean find(Node node, int data) {
    if(node == null) return false;

    if(data > node.data) {
        return find(node.right, data);
    } else if(data < node.data) {
        return find(node.left, data);
    } else {
        // data found
        return true;
    }
}
```

Add node in BST:→



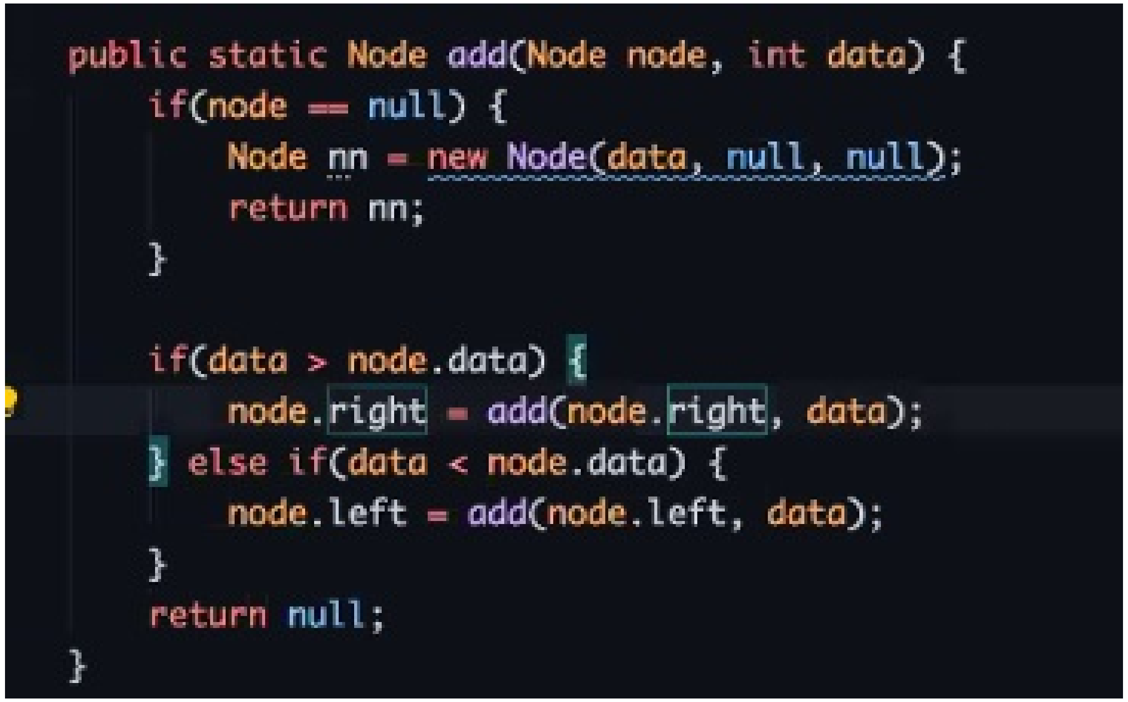
✓add → 87

✓add → 70

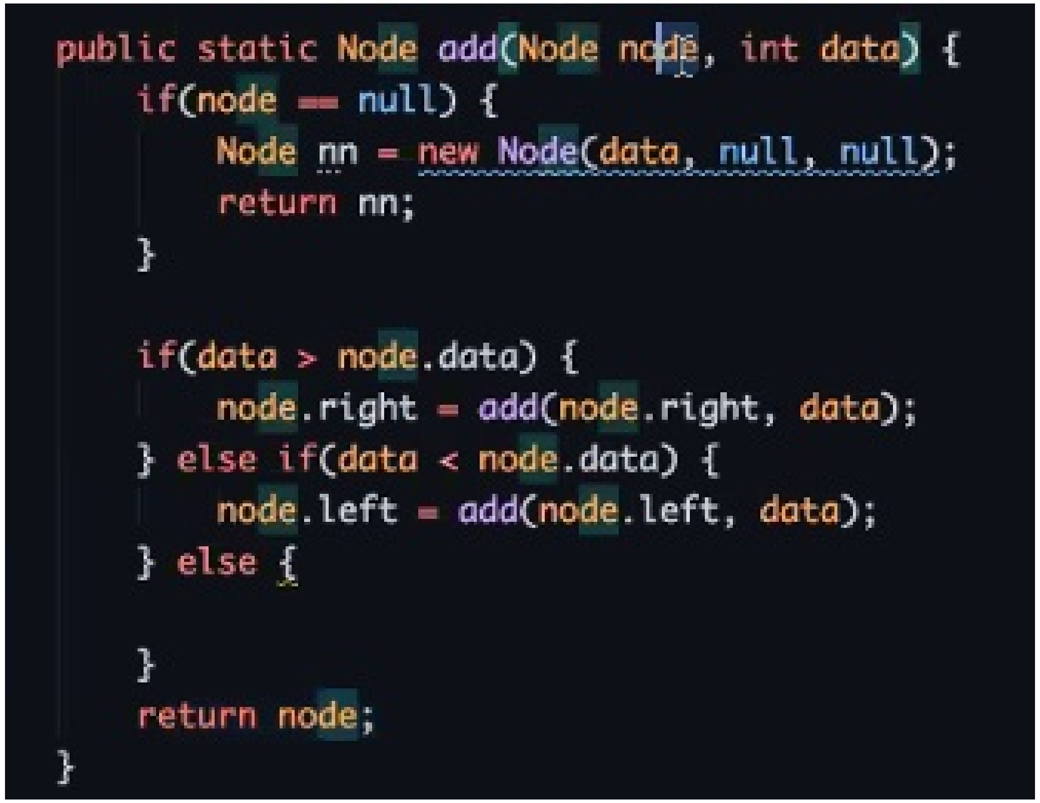
✓add → 20

✓add → 10

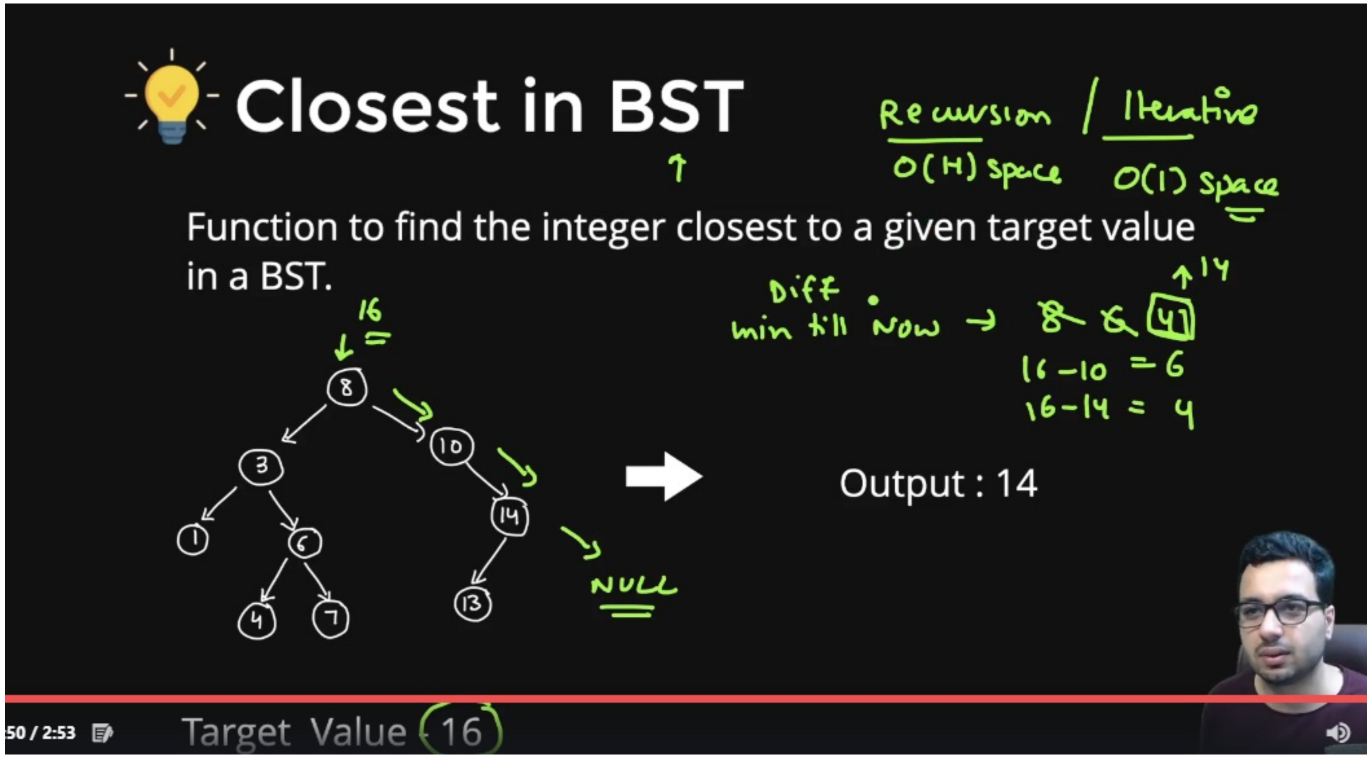
addNode is similar to find



```
public static Node add(Node node, int data) {
    if(node == null) {
        Node nn = new Node(data, null, null);
        return nn;
    }

    if(data > node.data) {
        node.right = add(node.right, data);
    } else if(data < node.data) {
        node.left = add(node.left, data);
    }
    return null;
}
```



```
public static Node add(Node node, int data) {
    if(node == null) {
        Node nn = new Node(data, null, null);
        return nn;
    }

    if(data > node.data) {
        node.right = add(node.right, data);
    } else if(data < node.data) {
        node.left = add(node.left, data);
    } else {

    }
    return node;
}
```



Closest in BST

Recursion / Iterative

O(H) space O(1) space

Function to find the integer closest to a given target value in a BST.

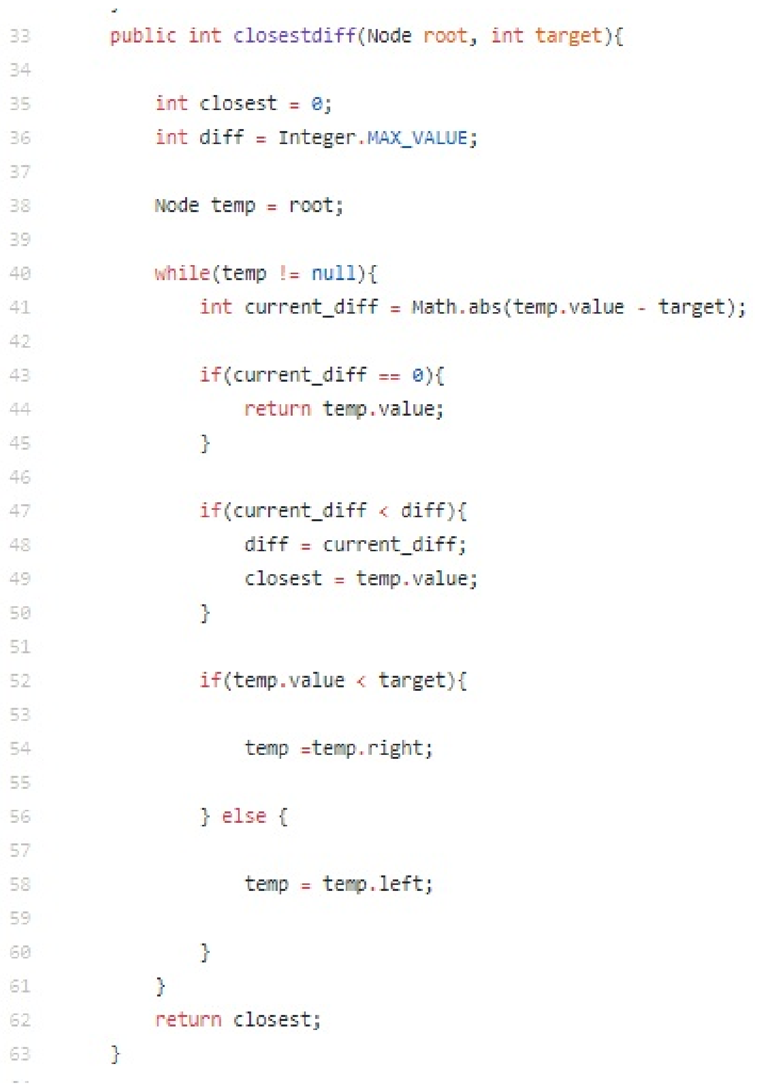
Diff min till now → 8 & 10 14

16-10 = 6

16-14 = 4

Output : 14

Target Value 16



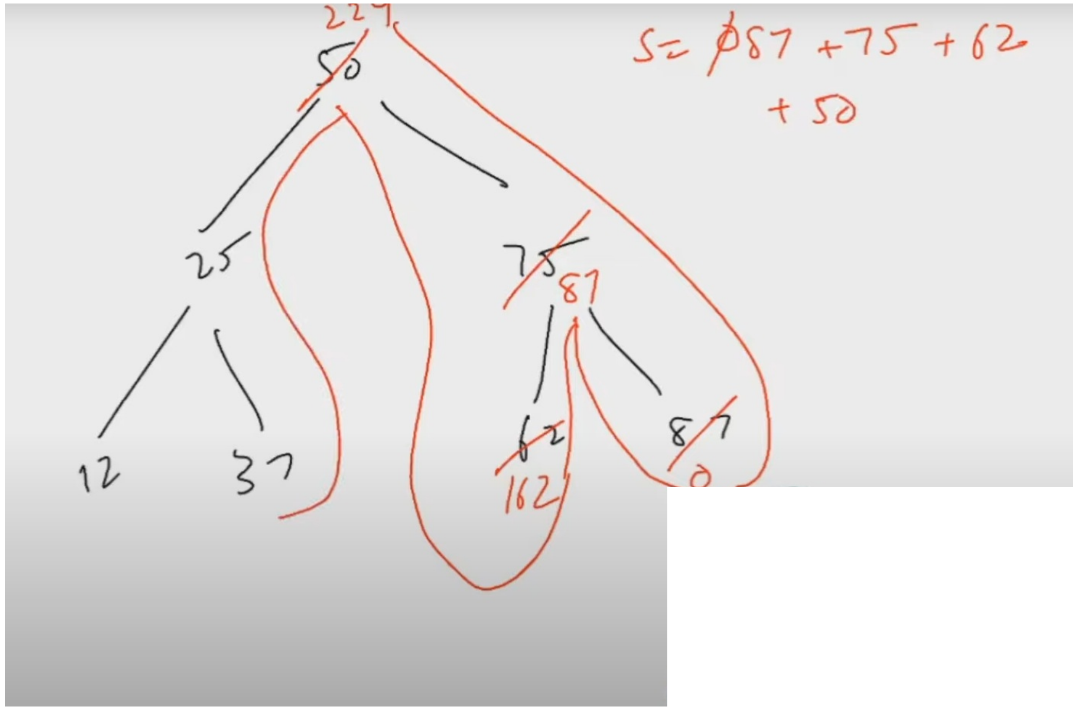
```java
public int closestdiff(Node root, int target){

    int closest = 0;
    int diff = Integer.MAX_VALUE;

    Node temp = root;

    while(temp != null){
        int current_diff = Math.abs(temp.value - target);

        if(current_diff == 0){
            return temp.value;
        }

        if(current_diff < diff){
            diff = current_diff;
            closest = temp.value;
        }

        if(temp.value < target){

            temp =temp.right;

        } else {

            temp = temp.left;

        }
    }
    return closest;
}
```

Replace Sum of its Larger Value

in bst , whne we traverse inorder, we get so
i.e inorder traversal in bst == sorted order t

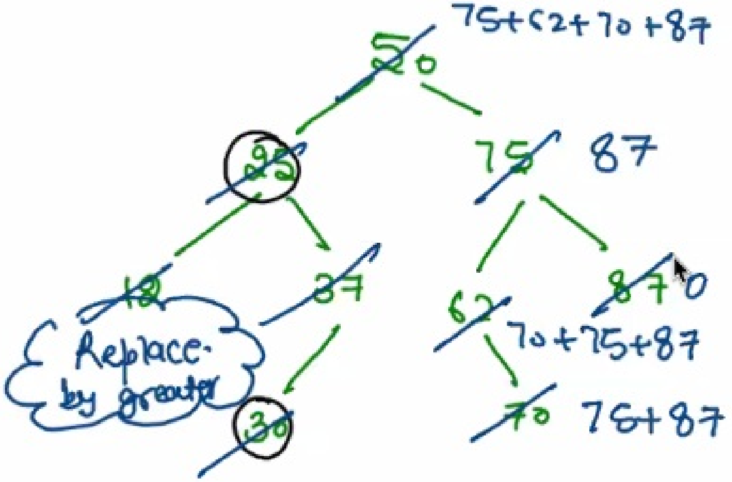
traverse in reveerse
right -> Node => left
so in this , larger nos will be traverseed first

take static sum,
replace node with sum
add node in sum;



Replace sum of its Larger value

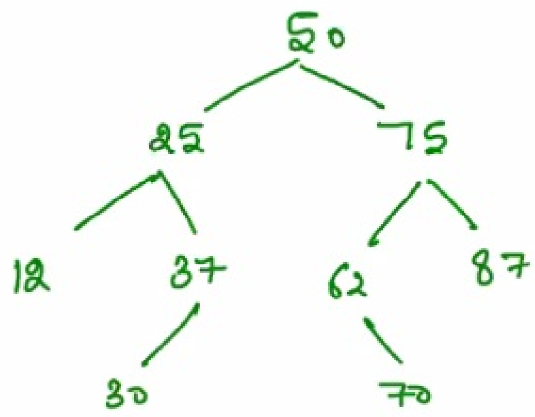
→ node data Replace by sum of it's larger value.



$\underset{X}{\cancel{25}} \rightarrow$ sum of its greater value = $\underbrace{80+37+50+62+70+75}_{x}+87$

Time Complexity → O(n)

Space : Recursive O(height)



Inorder→ Sorted Number

Reverse Inorder→ Decreasing order.

10 20 30 40 50

Sum less than value

0 10 30 60 100

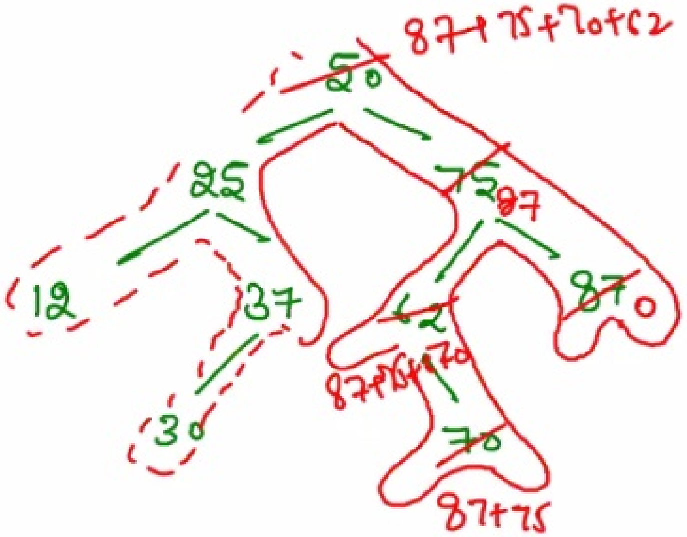
sum=~~0~~ ~~10~~ ~~30~~ ~~60~~ ~~100~~ 150

140 120 90 50 0

Sum greater than value

sum=~~0~~ ~~50~~ ~~90~~ ~~120~~ ~~140~~ 150

Tree.



Inorder → Increasing order

Reverse InOrder → Decreasing order

sum = 0 + 87 + 75 + 70 + 62 + 50

Traverse - Reverse Inorder

⓪ - get data.

Step. ① Replace value by sum

② provide impact of data on sum.

3 - h

Example.

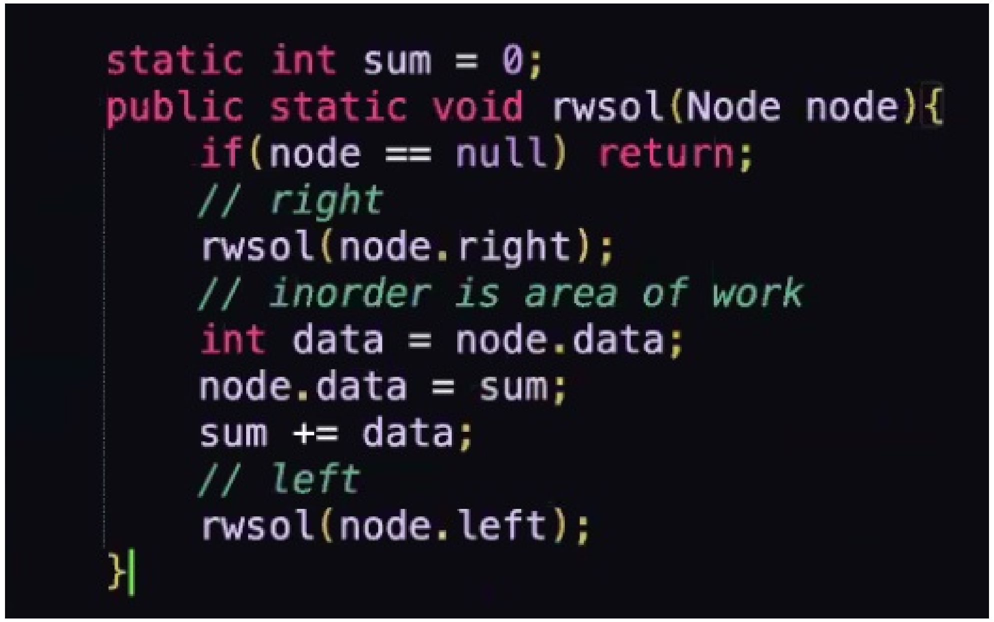
arr → 10 20 30 40 50

new values ⇒ 0 10 30 60 100

replace value by sum less than data = arr[i]   sum ⇒ ~~0~~ ~~10~~ ~~30~~ ~~60~~ ~~100~~ 150

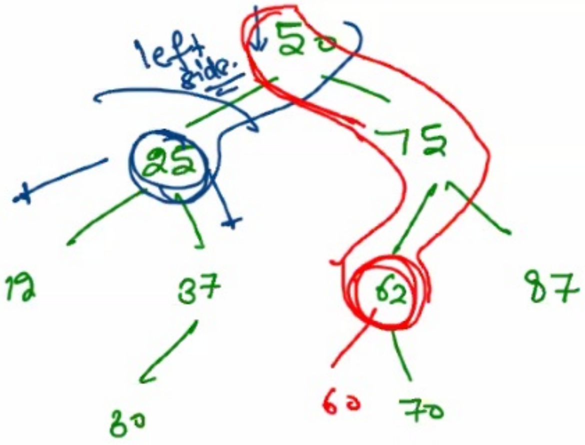
replace value by sum greater than value = arr[i]   Values = 140 120 90 50 0

Sum = ~~0~~ ~~50~~ ~~90~~ ~~120~~ ~~140~~ 150



```java
static int sum = 0;
public static void rwsol(Node node){
    if(node == null) return;
    // right
    rwsol(node.right);
    // inorder is area of work
    int data = node.data;
    node.data = sum;
    sum += data;
    // left
    rwsol(node.left);
}
```

① LCA of BST



12, 87

60 - 70

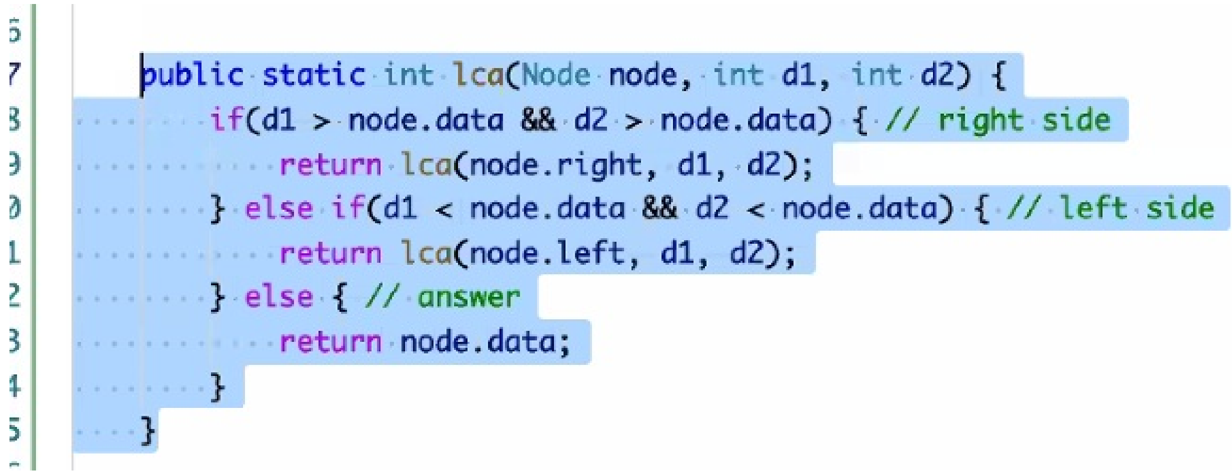
25

62

Case-I   d1 > node.data   d2 > no.data.
→ right side,

Case II   d1 < node.data   d2 < node.data
→ left side,

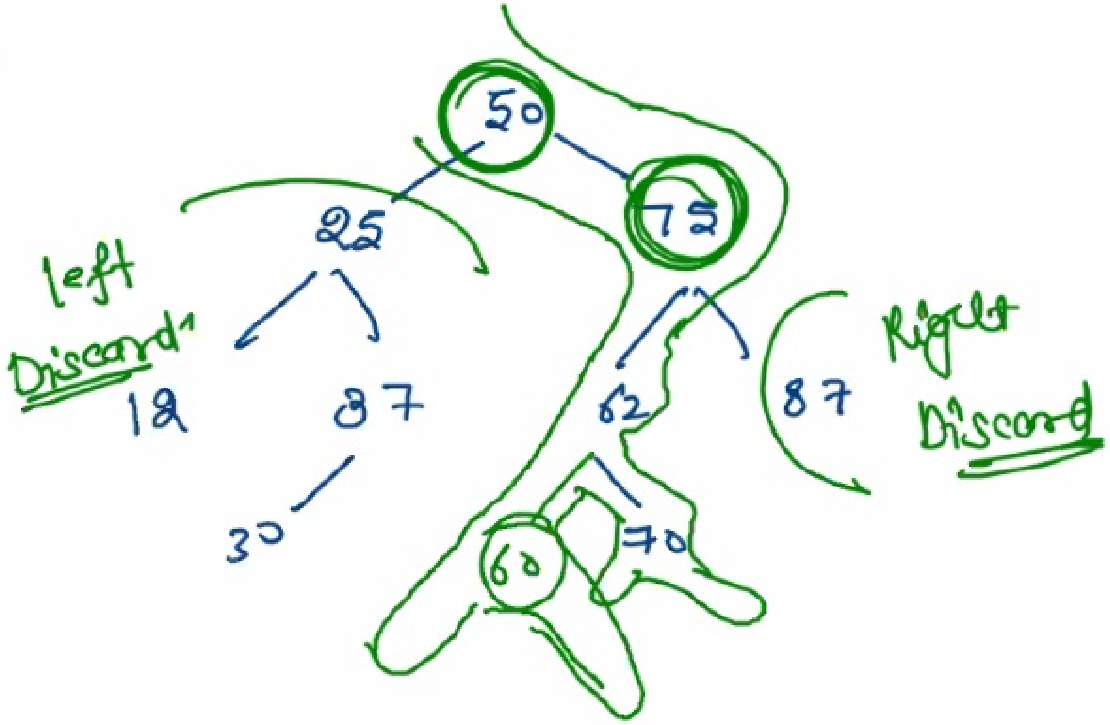
Case-III   splitting -   node.data is LCA   print
& return.



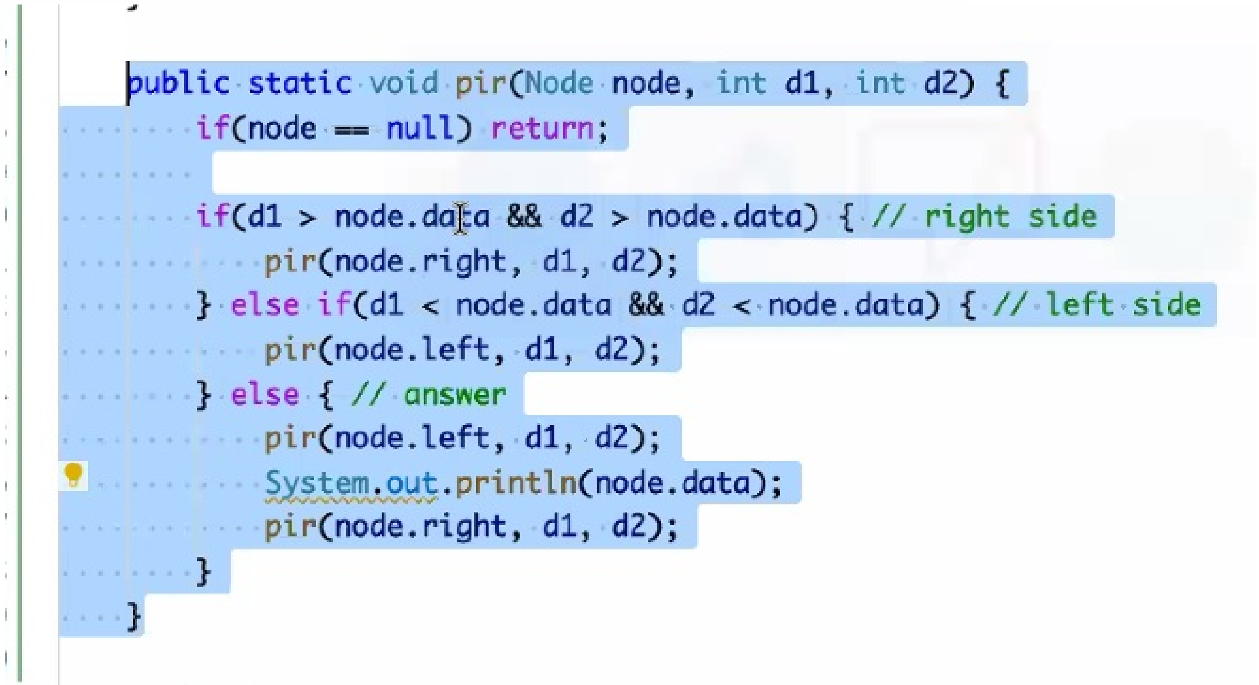
```
    public static int lca(Node node, int d1, int d2) {
        if(d1 > node.data && d2 > node.data) { // right side
            return lca(node.right, d1, d2);
        } else if(d1 < node.data && d2 < node.data) { // left side
            return lca(node.left, d1, d2);
        } else { // answer
            return node.data;
        }
    }
```

Print in Range

60 73



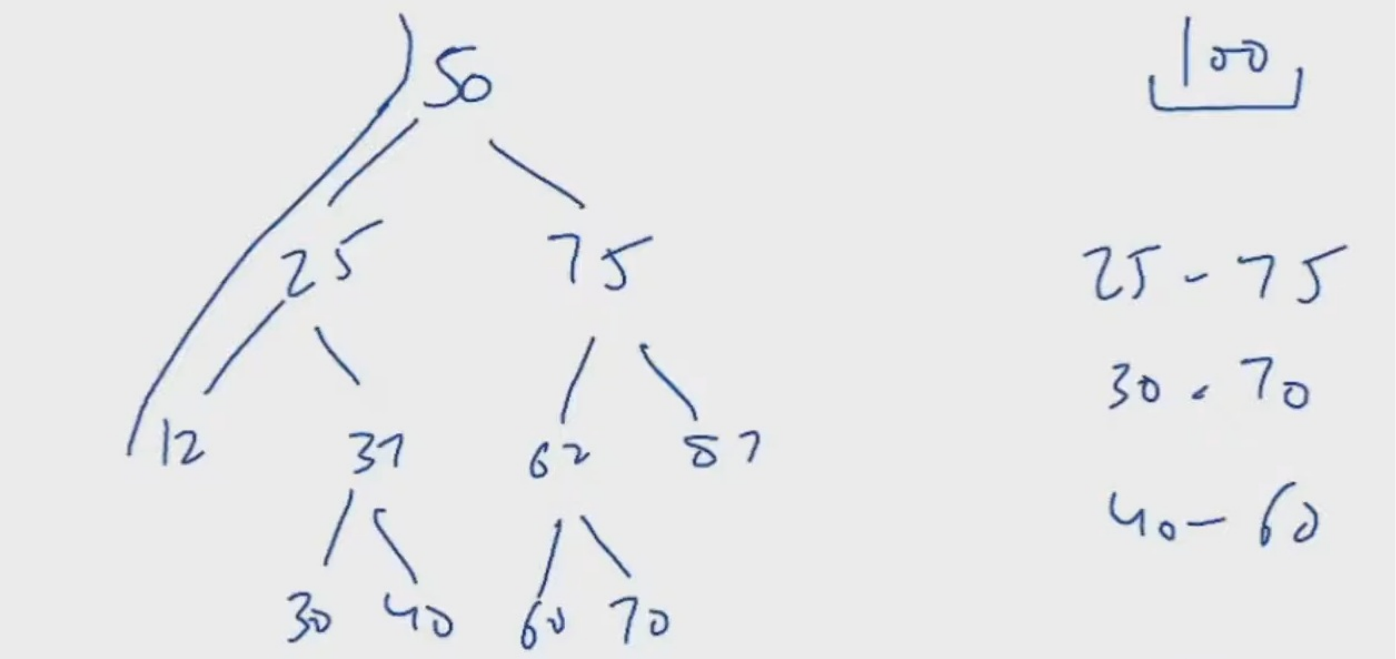
60 62 70



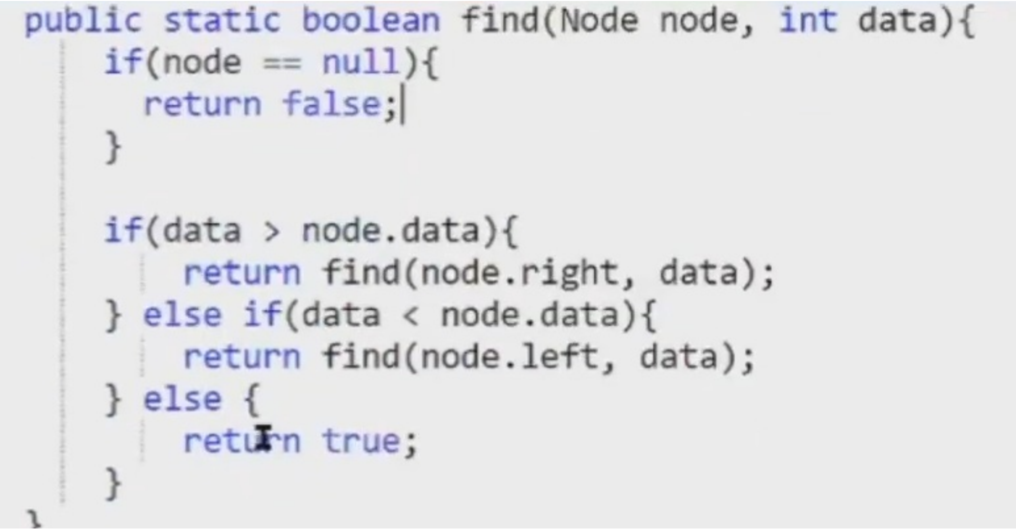
```java
public static void pir(Node node, int d1, int d2) {
    if(node == null) return;

    if(d1 > node.data && d2 > node.data) { // right side
        pir(node.right, d1, d2);
    } else if(d1 < node.data && d2 < node.data) { // left side
        pir(node.left, d1, d2);
    } else { // answer
        pir(node.left, d1, d2);
        System.out.println(node.data);
        pir(node.right, d1, d2);
    }
}
```

Target sum pair

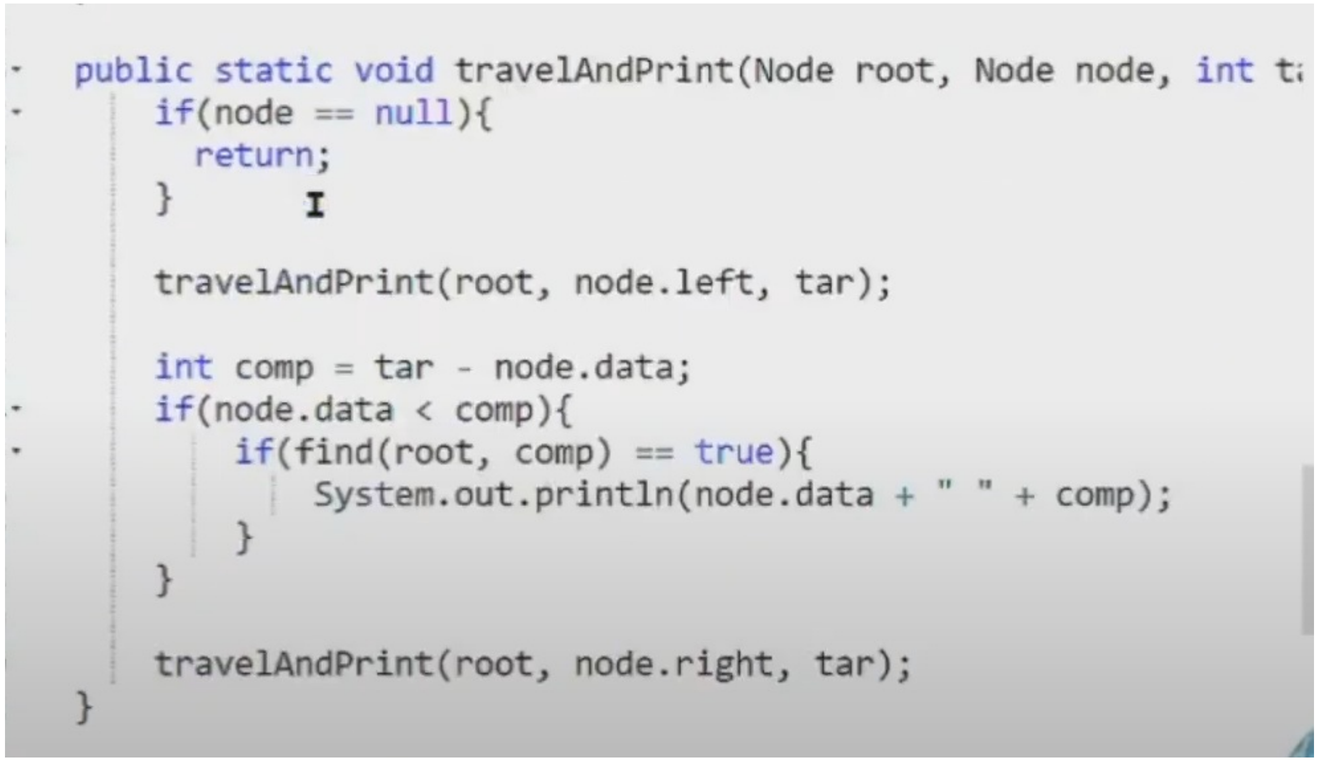
	Time.	Space.
Method 1	nh traversal + find	h (Recursive)
method 2 Ins. → left pointer → right pointer.	n	n Arraylist fill → In Area Sorted arraylist
method3	n better	h better



approach 1



```
public static boolean find(Node node, int data){
    if(node == null){
      return false;
    }

    if(data > node.data){
        return find(node.right, data);
    } else if(data < node.data){
        return find(node.left, data);
    } else {
        return true;
    }
}
```

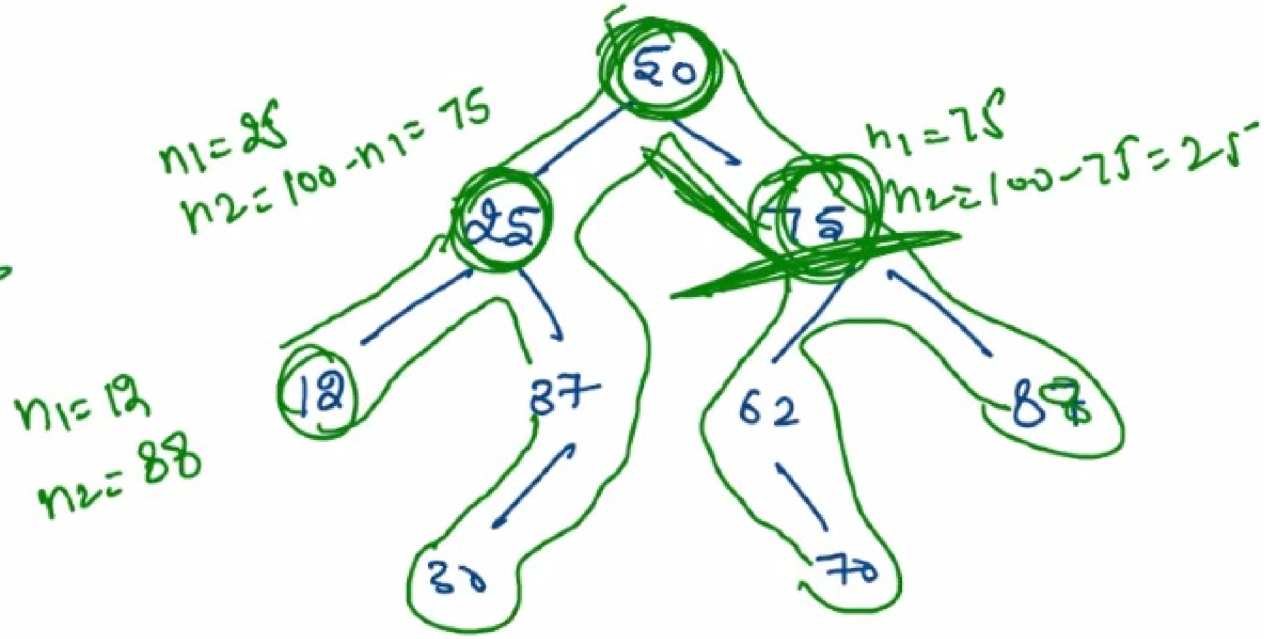


```
public static void travelAndPrint(Node root, Node node, int t
    if(node == null){
      return;
    }

    travelAndPrint(root, node.left, tar);

    int comp = tar - node.data;
    if(node.data < comp){
        if(find(root, comp) == true){
            System.out.println(node.data + " " + comp);
        }
    }

    travelAndPrint(root, node.right, tar);
}
```

25 75

12 88

30 70



12 88

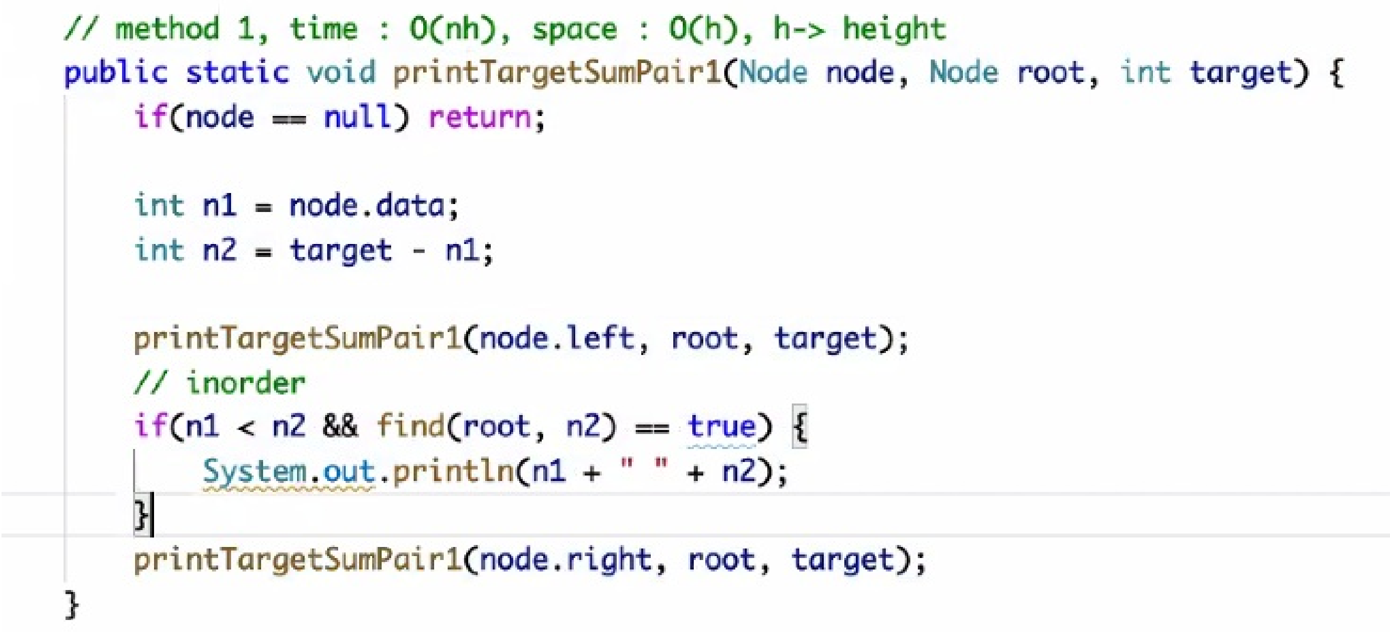
25 75

30 70

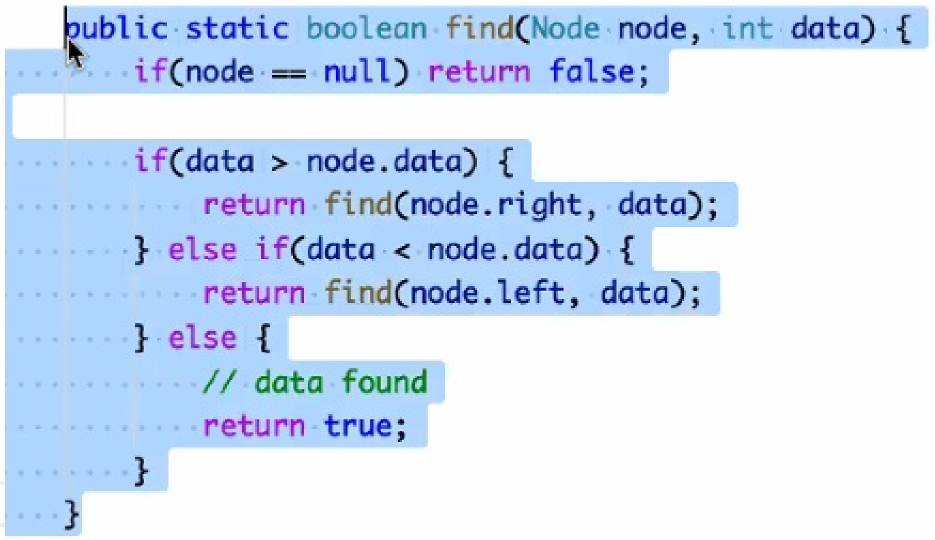
Method 1

nh
traversal + find

h (Recursive)



```
// method 1, time : O(nh), space : O(h), h-> height
public static void printTargetSumPair1(Node node, Node root, int target) {
    if(node == null) return;

    int n1 = node.data;
    int n2 = target - n1;

    printTargetSumPair1(node.left, root, target);
    // inorder
    if(n1 < n2 && find(root, n2) == true) {
        System.out.println(n1 + " " + n2);
    }
    printTargetSumPair1(node.right, root, target);
}
```



```
public static boolean find(Node node, int data) {
    if(node == null) return false;

    if(data > node.data) {
        return find(node.right, data);
    } else if(data < node.data) {
        return find(node.left, data);
    } else {
        // data found
        return true;
    }
}
```

method 2

tns:

→ left pointer

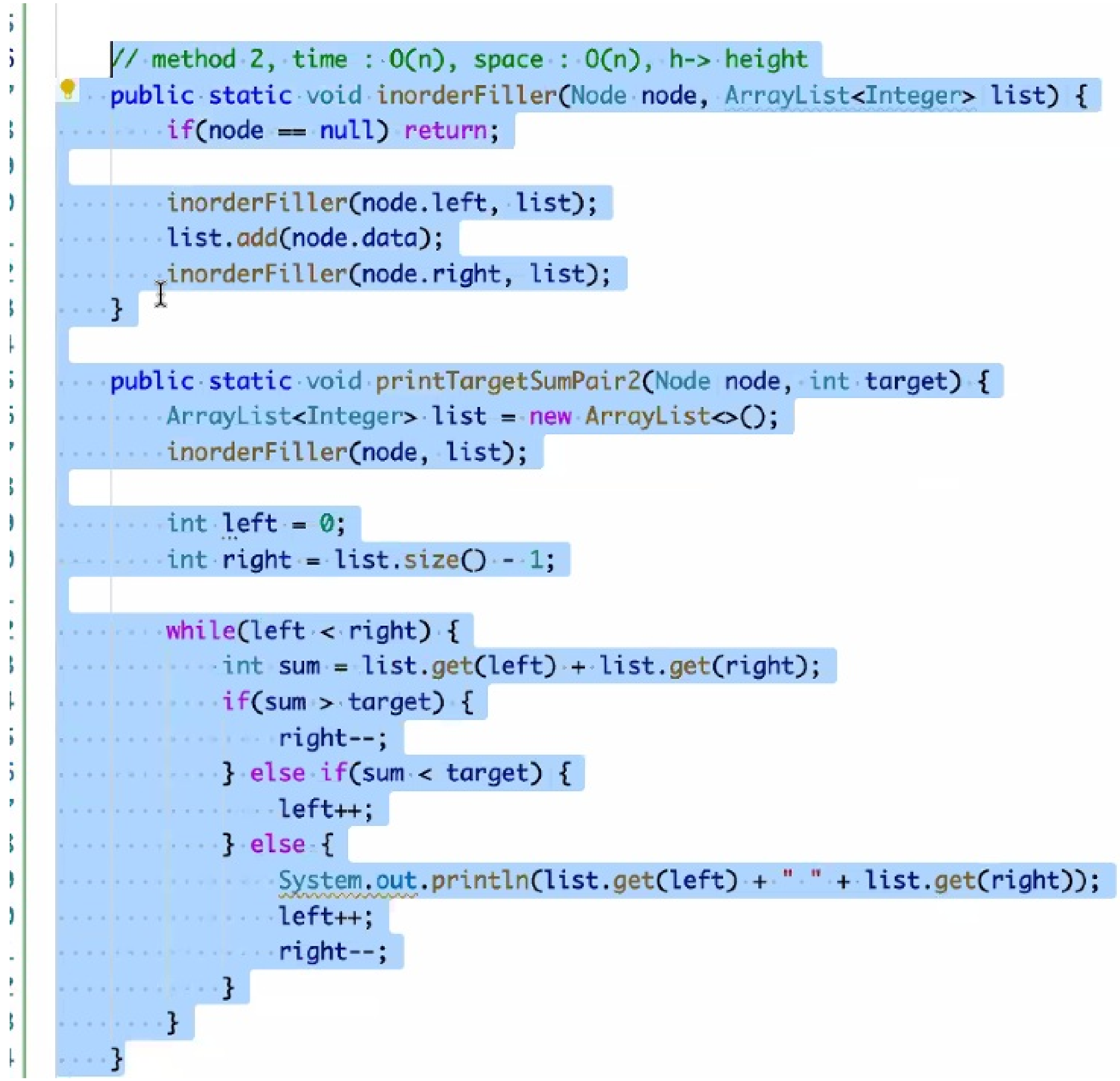
→ right pointer

n

n

Arraylist fill → In Area

Sorted arraylist

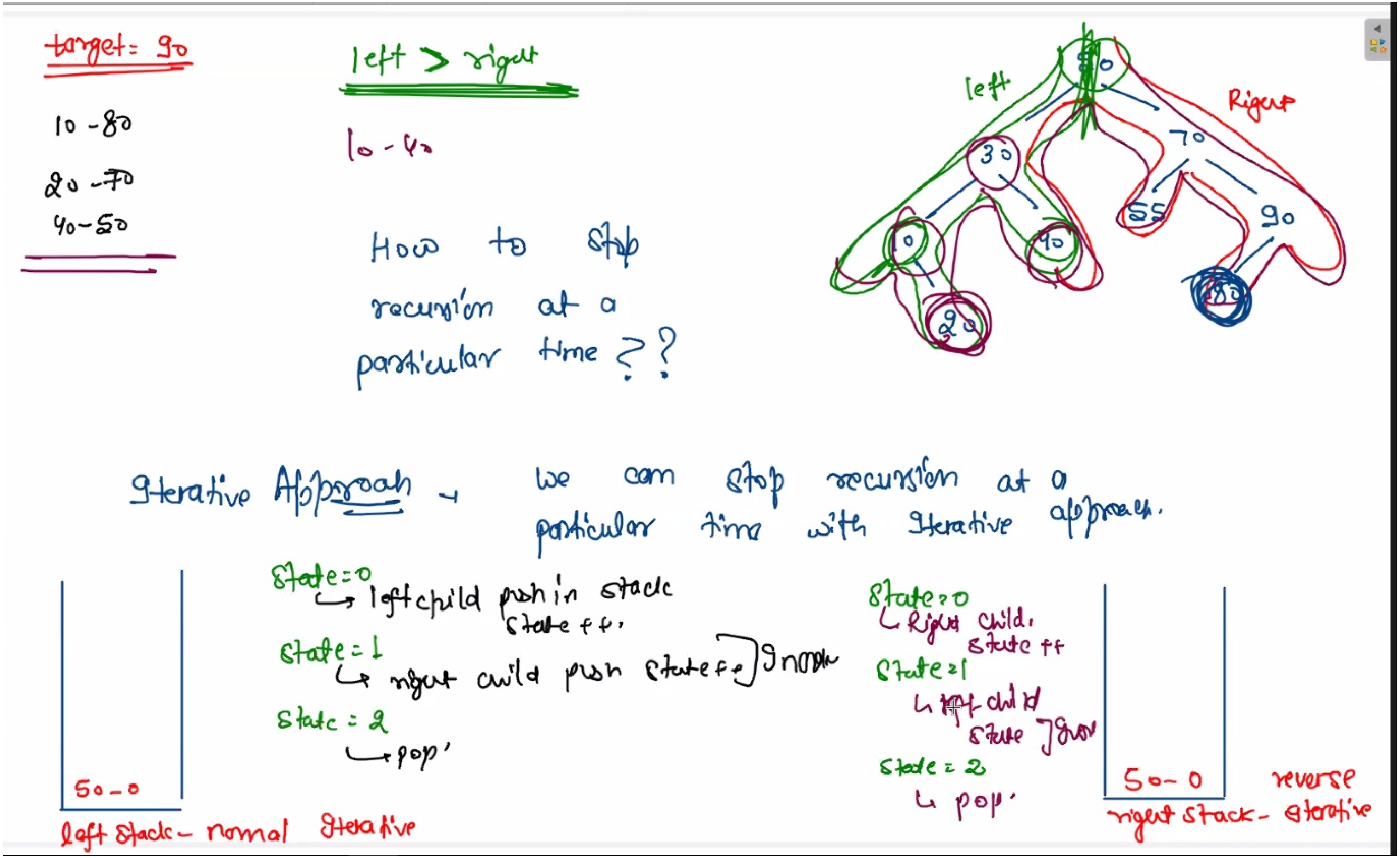


```
    // method 2, time : O(n), space : O(n), h-> height
    public static void inorderFiller(Node node, ArrayList<Integer> list) {
        if(node == null) return;

        inorderFiller(node.left, list);
        list.add(node.data);
        inorderFiller(node.right, list);
    }

    public static void printTargetSumPair2(Node node, int target) {
        ArrayList<Integer> list = new ArrayList<>();
        inorderFiller(node, list);

        int left = 0;
        int right = list.size() - 1;

        while(left < right) {
            int sum = list.get(left) + list.get(right);
            if(sum > target) {
                right--;
            } else if(sum < target) {
                left++;
            } else {
                System.out.println(list.get(left) + " " + list.get(right));
                left++;
                right--;
            }
        }
    }
```

method3

n

h



target: 90

10 - 80

20 - 70

40 - 50

left > right

10 - 40

How to stop recursion at a particular time ??

Iterative Approach → we can stop recursion at a particular time with Iterative approach.

State=0
→ leftchild push in stack state++.

State=1
→ right child push state++] Inorder

State = 2
→ pop

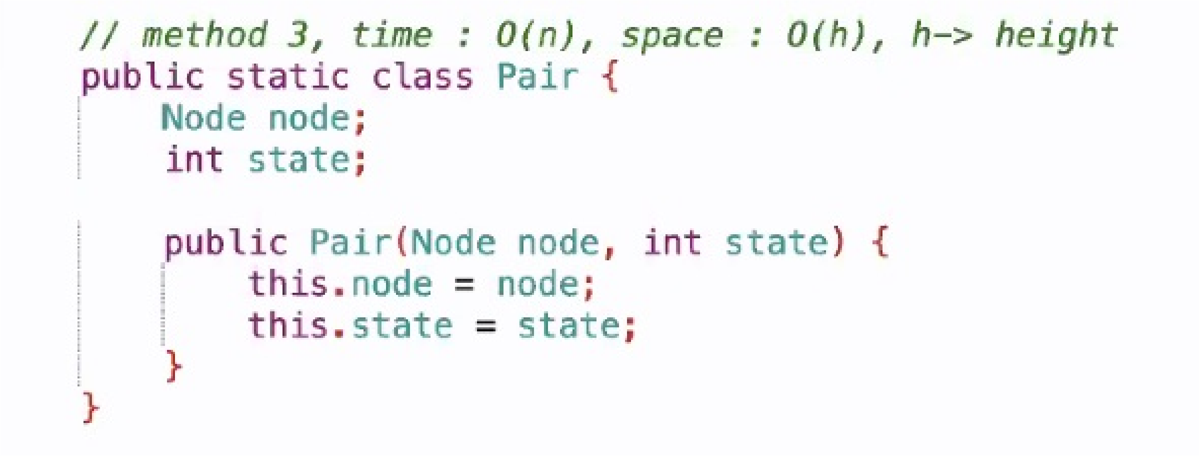
State=0
↳ Right child, state ++

State=1
↳ left child state] Inorder

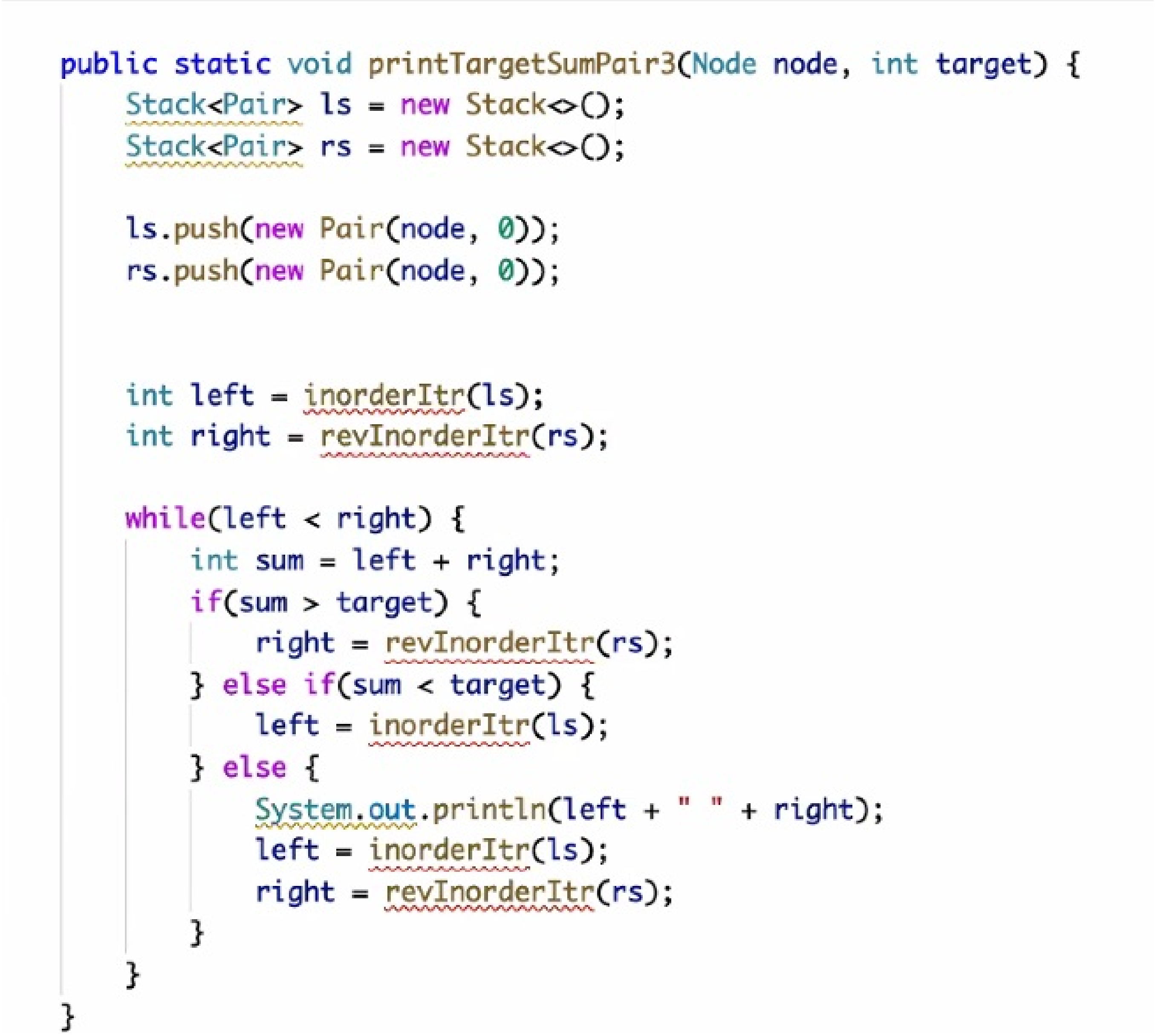
State = 2
↳ pop

left stack - normal Iterative

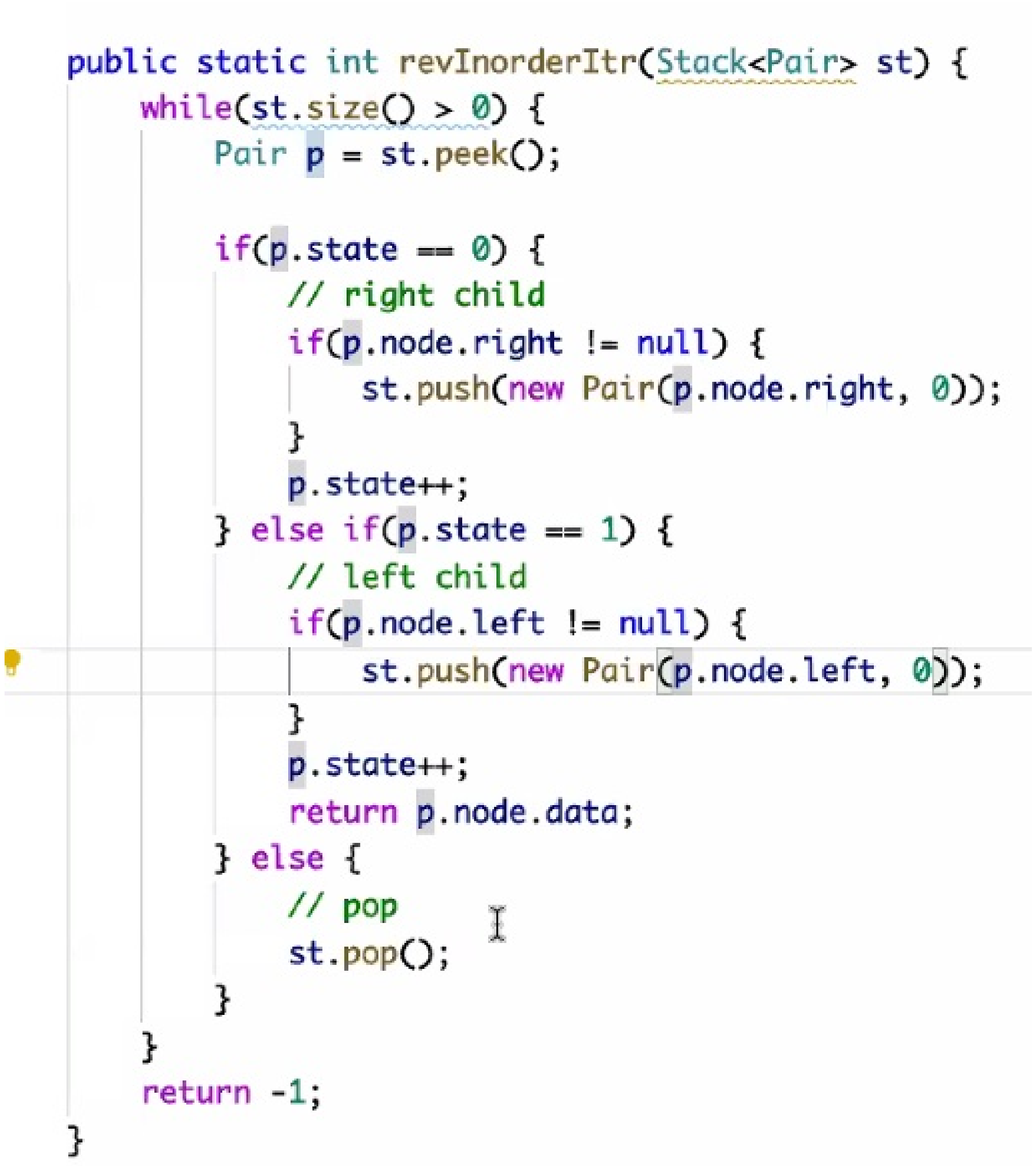
right stack - Iterative reverse



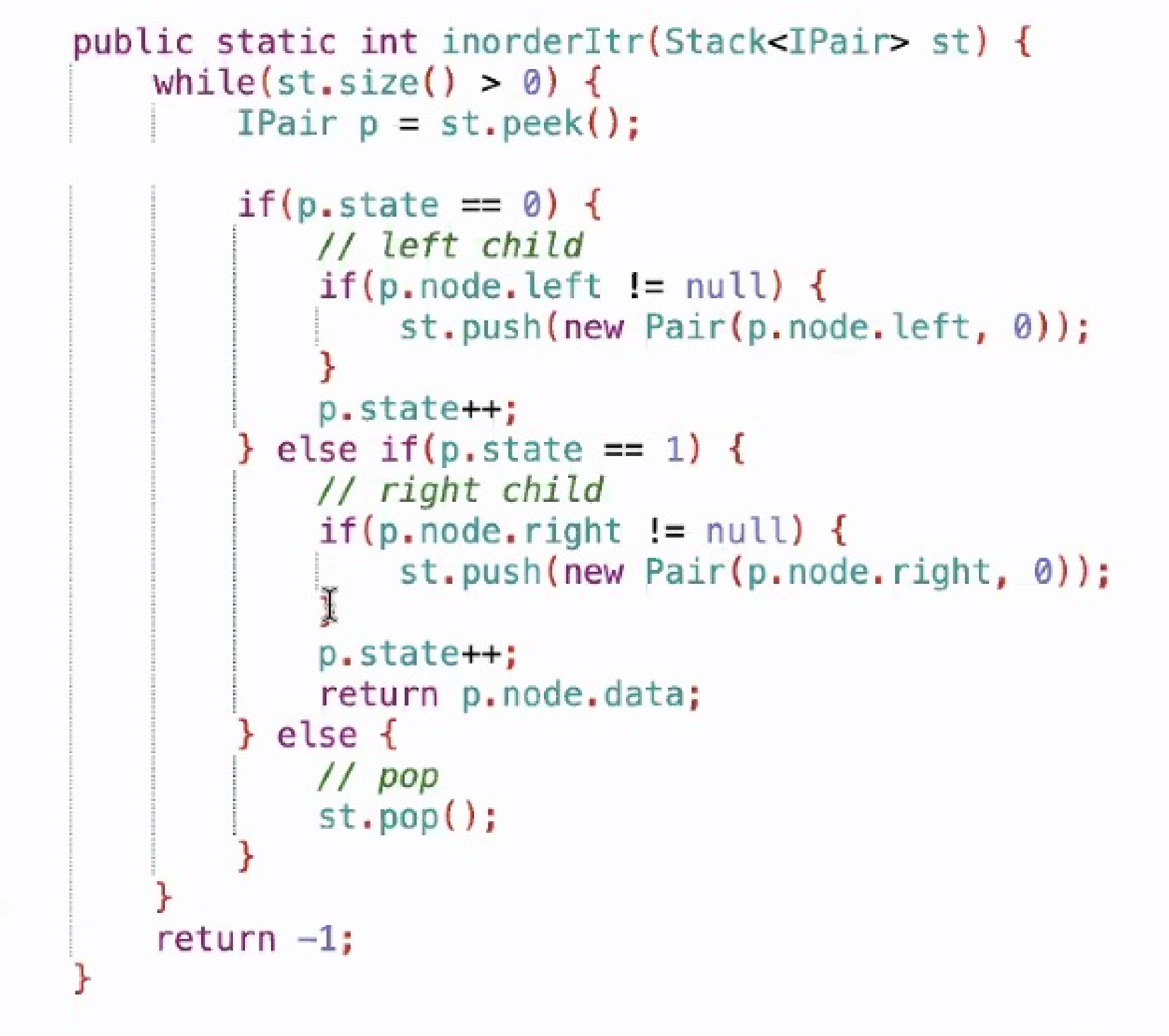
```
// method 3, time : O(n), space : O(h), h-> height
public static class Pair {
    Node node;
    int state;

    public Pair(Node node, int state) {
        this.node = node;
        this.state = state;
    }
}
```



```
public static void printTargetSumPair3(Node node, int target) {
    Stack<Pair> ls = new Stack<>();
    Stack<Pair> rs = new Stack<>();

    ls.push(new Pair(node, 0));
    rs.push(new Pair(node, 0));


    int left = inorderItr(ls);
    int right = revInorderItr(rs);

    while(left < right) {
        int sum = left + right;
        if(sum > target) {
            right = revInorderItr(rs);
        } else if(sum < target) {
            left = inorderItr(ls);
        } else {
            System.out.println(left + " " + right);
            left = inorderItr(ls);
            right = revInorderItr(rs);
        }
    }
}
```



```java
public static int revInorderItr(Stack<Pair> st) {
    while(st.size() > 0) {
        Pair p = st.peek();

        if(p.state == 0) {
            // right child
            if(p.node.right != null) {
                st.push(new Pair(p.node.right, 0));
            }
            p.state++;
        } else if(p.state == 1) {
            // left child
            if(p.node.left != null) {
                st.push(new Pair(p.node.left, 0));
            }
            p.state++;
            return p.node.data;
        } else {
            // pop
            st.pop();
        }
    }
    return -1;
}
```



```java
public static int inorderItr(Stack<IPair> st) {
    while(st.size() > 0) {
        IPair p = st.peek();

        if(p.state == 0) {
            // left child
            if(p.node.left != null) {
                st.push(new Pair(p.node.left, 0));
            }
            p.state++;
        } else if(p.state == 1) {
            // right child
            if(p.node.right != null) {
                st.push(new Pair(p.node.right, 0));
            }
            p.state++;
            return p.node.data;
        } else {
            // pop
            st.pop();
        }
    }
    return -1;
}
```

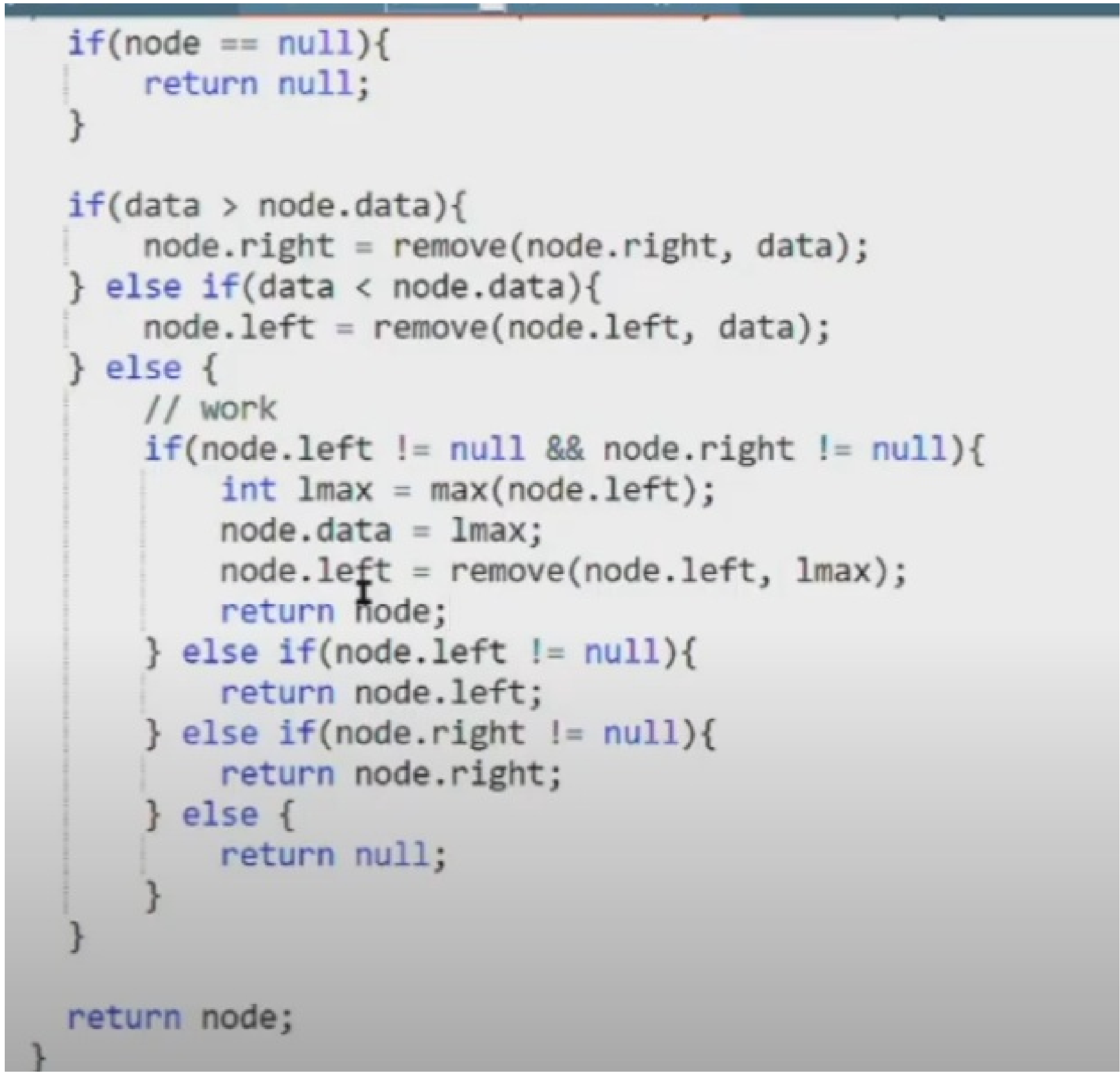
remove node in bst

the node you are removing can have
either 0 child
or 1 child
or 2 child

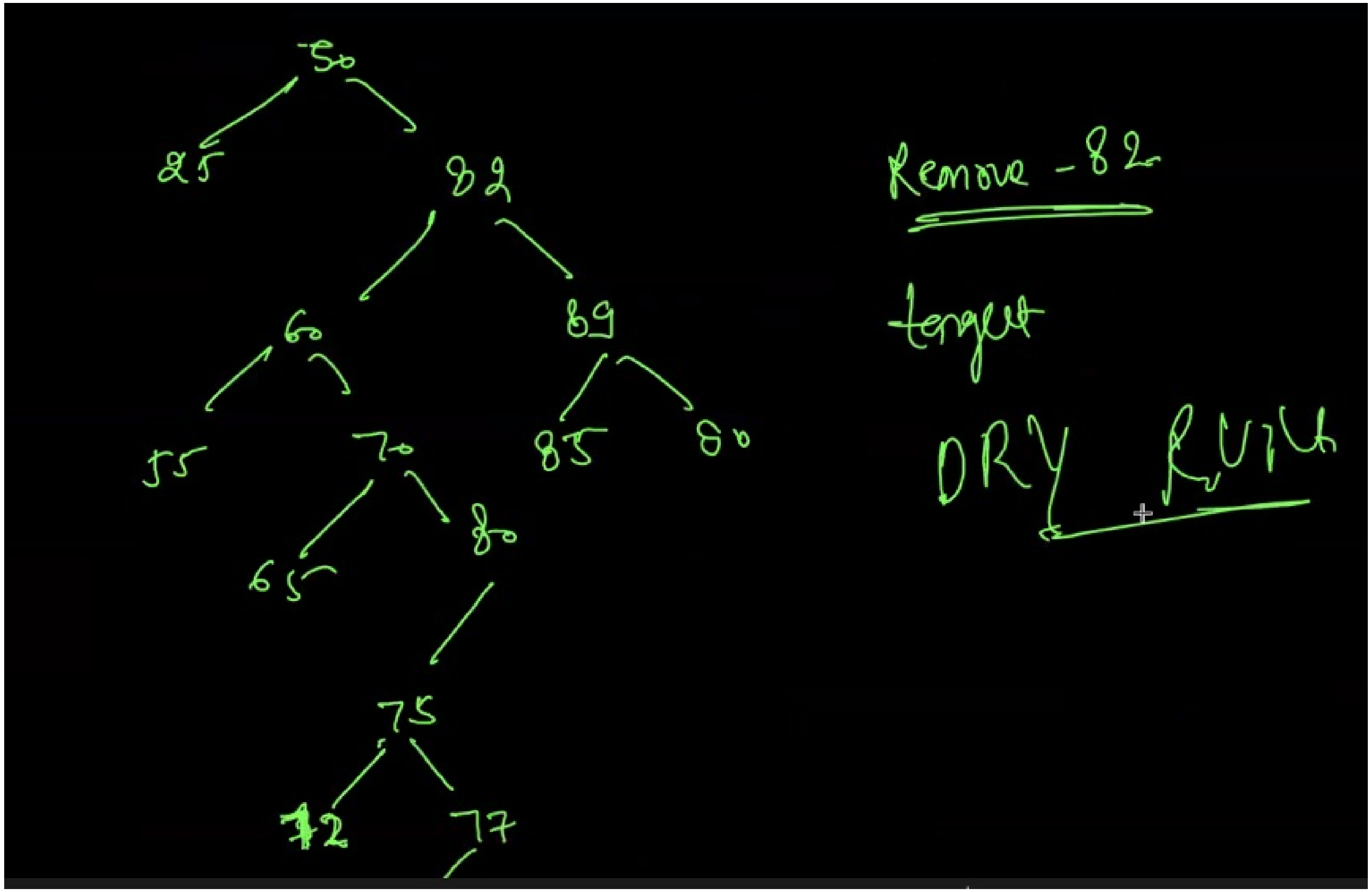
if 0 child , return null;

if one child, return node.left / node.right

if 2 child, get left max, set node data, remove lmax



```
    if(node == null){
        return null;
    }

    if(data > node.data){
        node.right = remove(node.right, data);
    } else if(data < node.data){
        node.left = remove(node.left, data);
    } else {
        // work
        if(node.left != null && node.right != null){
            int lmax = max(node.left);
            node.data = lmax;
            node.left = remove(node.left, lmax);
            return node;
        } else if(node.left != null){
            return node.left;
        } else if(node.right != null){
            return node.right;
        } else {
            return null;
        }
    }

    return node;
}
```



Remove - 82

target

DRY RUN