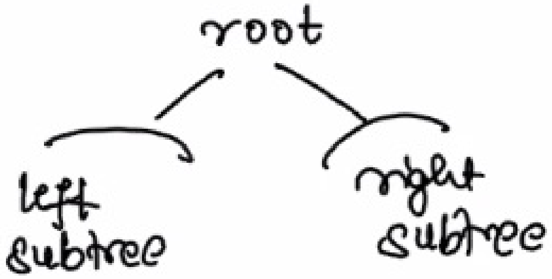
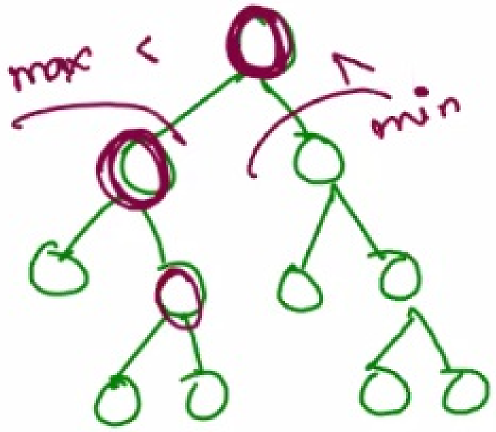


for BST

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

all nodes in right subtree is greater tham root.data

→ these are valid for all node in BST.



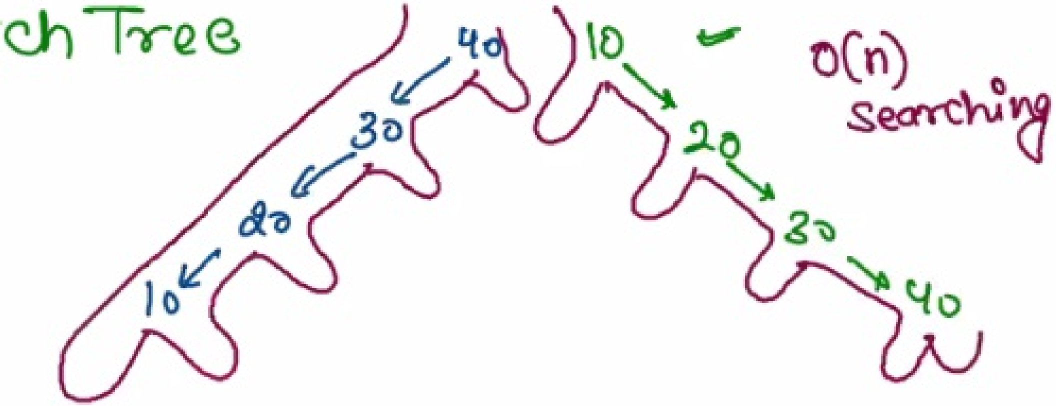
Benefits. → * searching.] Based on searching.

* store

* value based problem.

BST → Binary Search Tree

AVL → Balanced BST



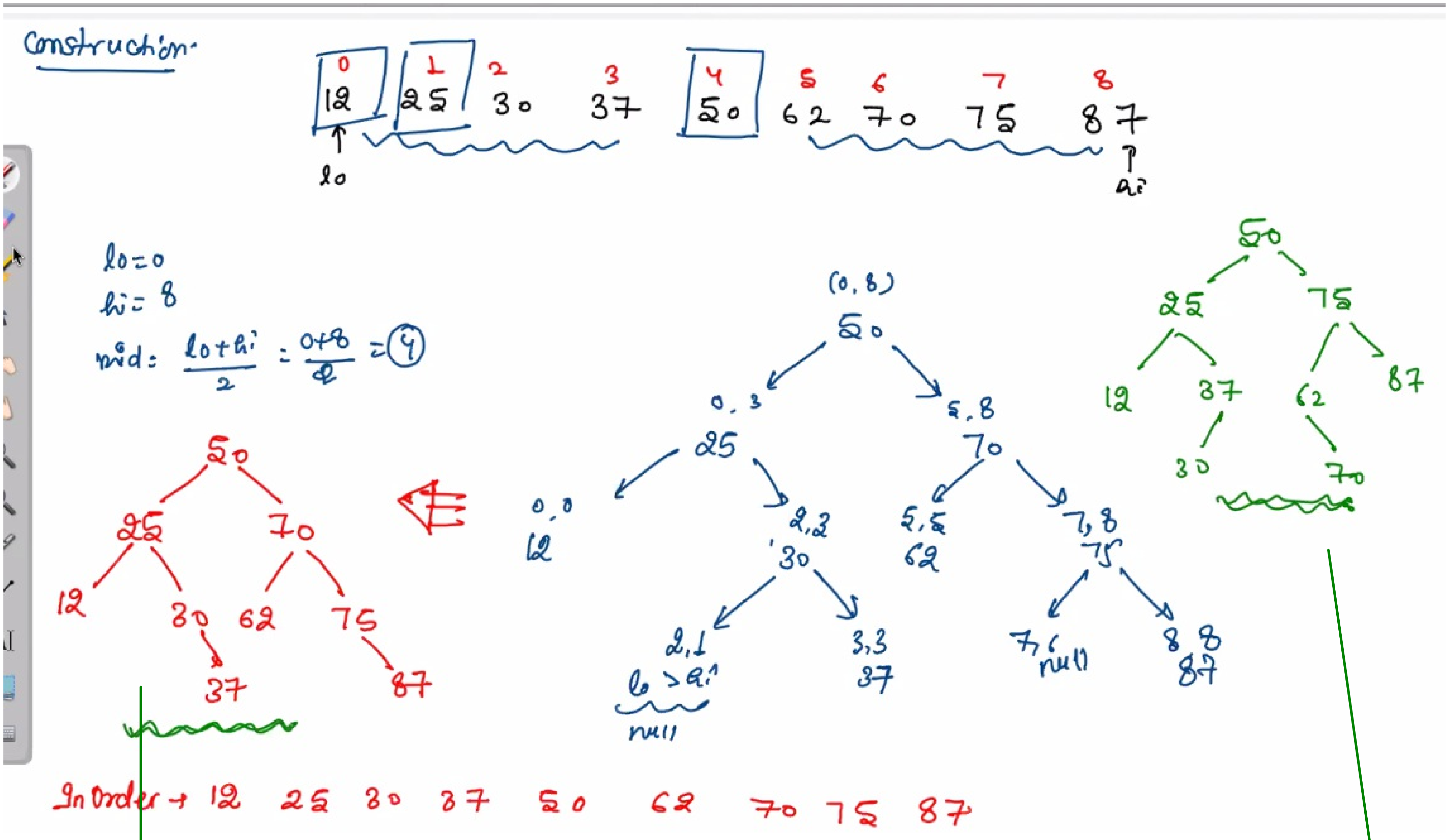
O(n) searching

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 ↑   ↑ hi

lo = 0

hi = 8

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

Inorder → 12 25 30 37 50 62 70 75 87

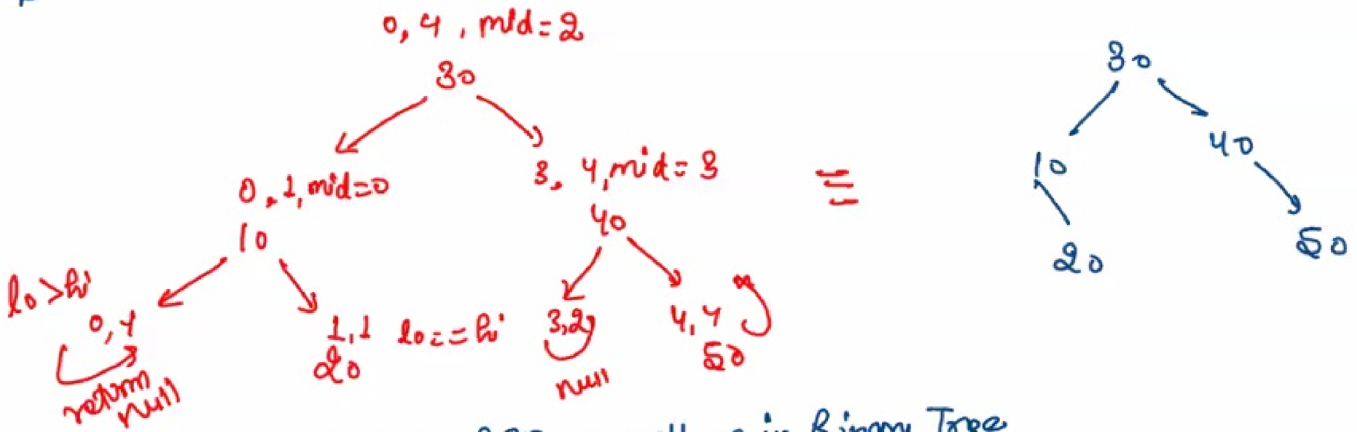
indorder

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

indorder

	0	1	2	3	4
data →	10	20	30	40	50

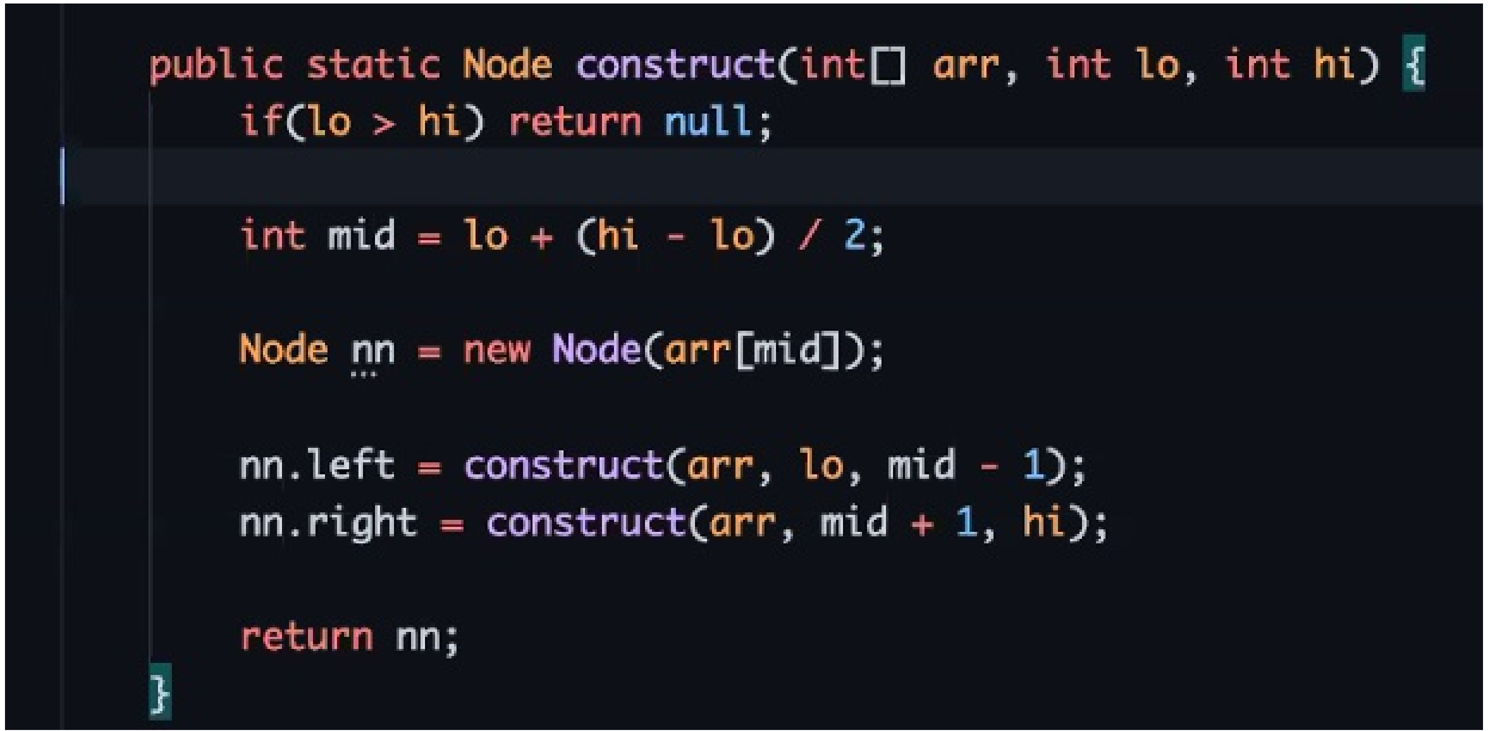
construction →



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

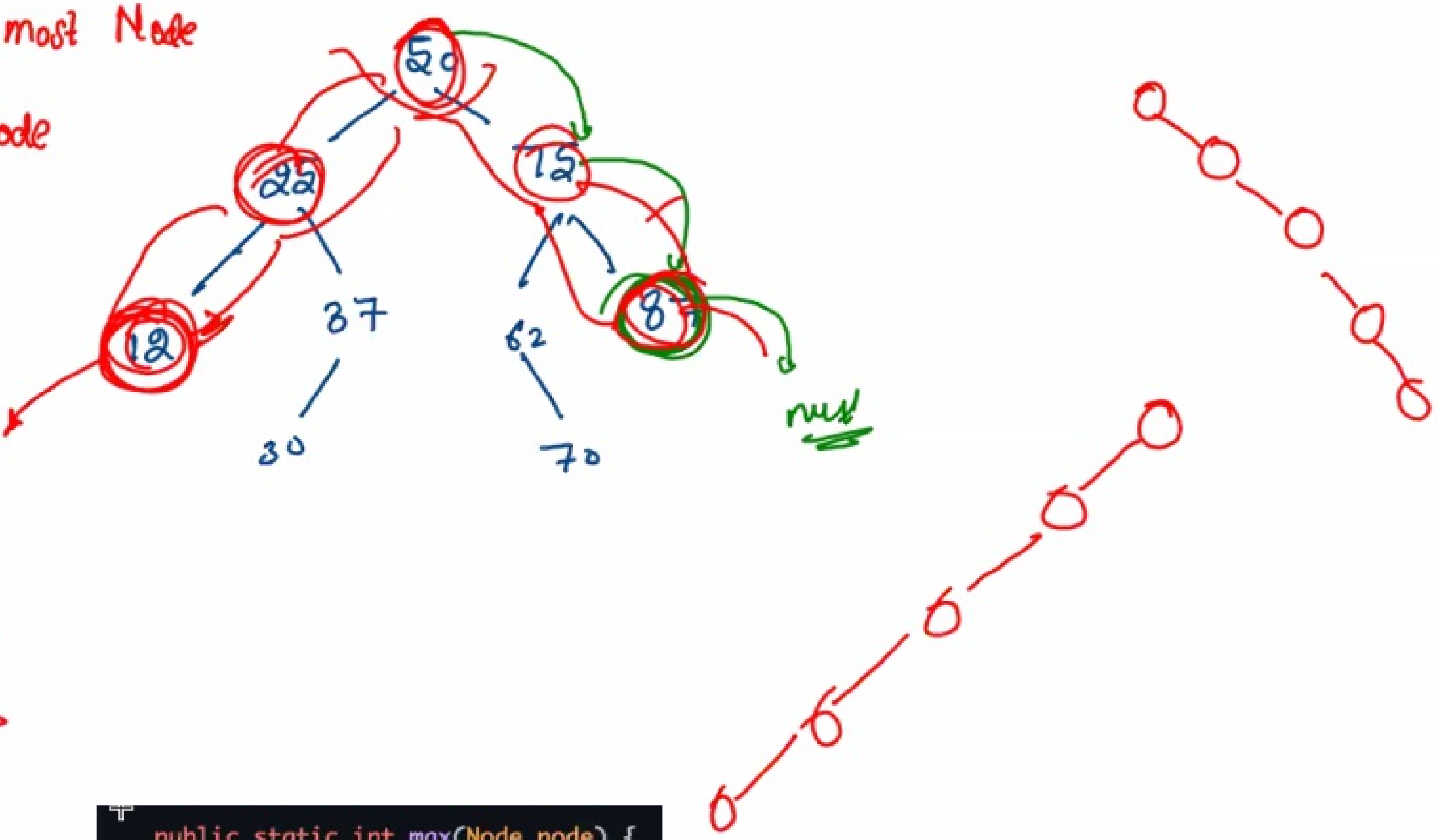
min → left most Node

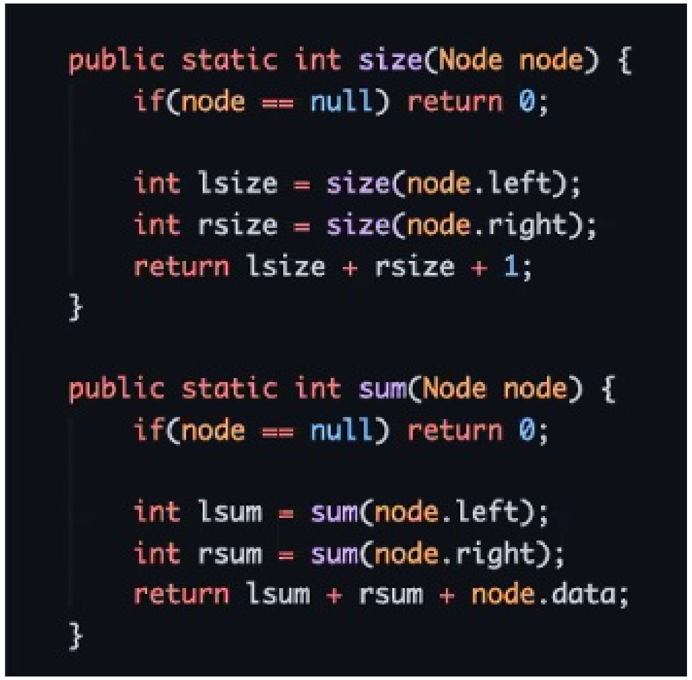
O(h)

h is height

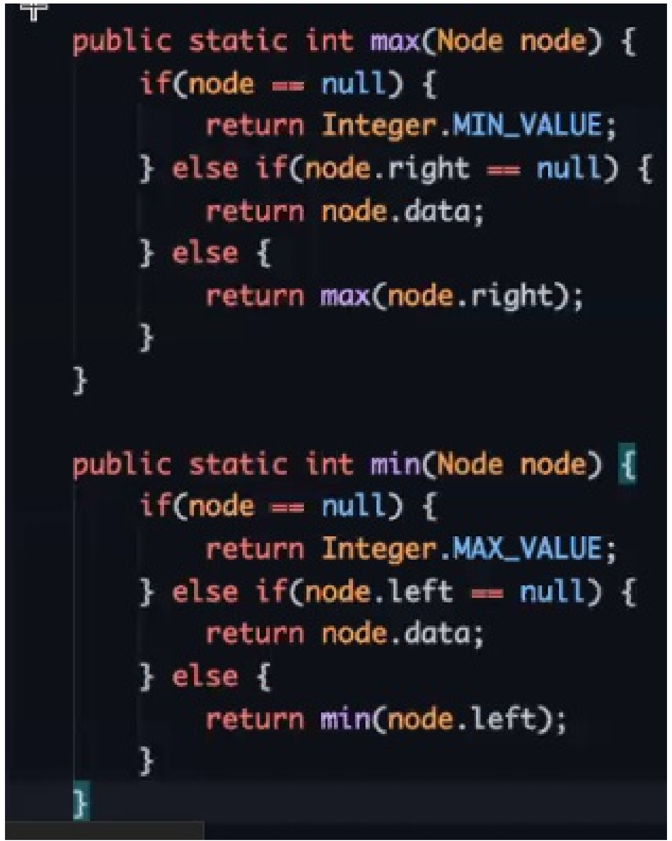
AVL → logn.

height = logn.

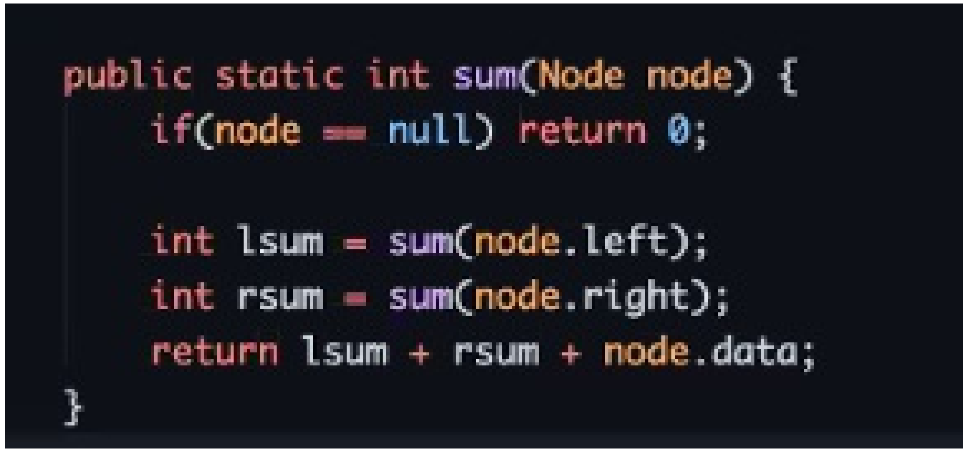




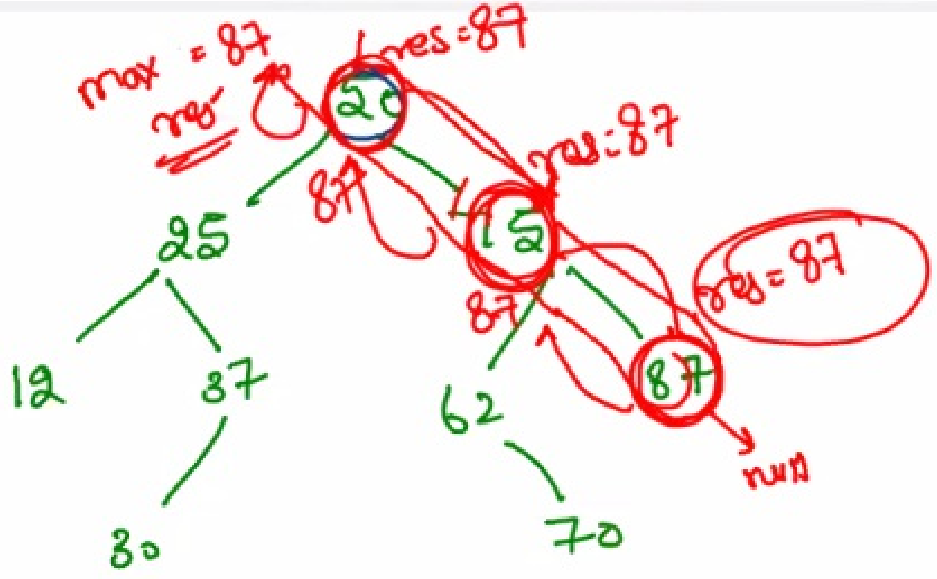
```
public static int size(Node node) {
    if(node == null) return 0;

    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;
}
```

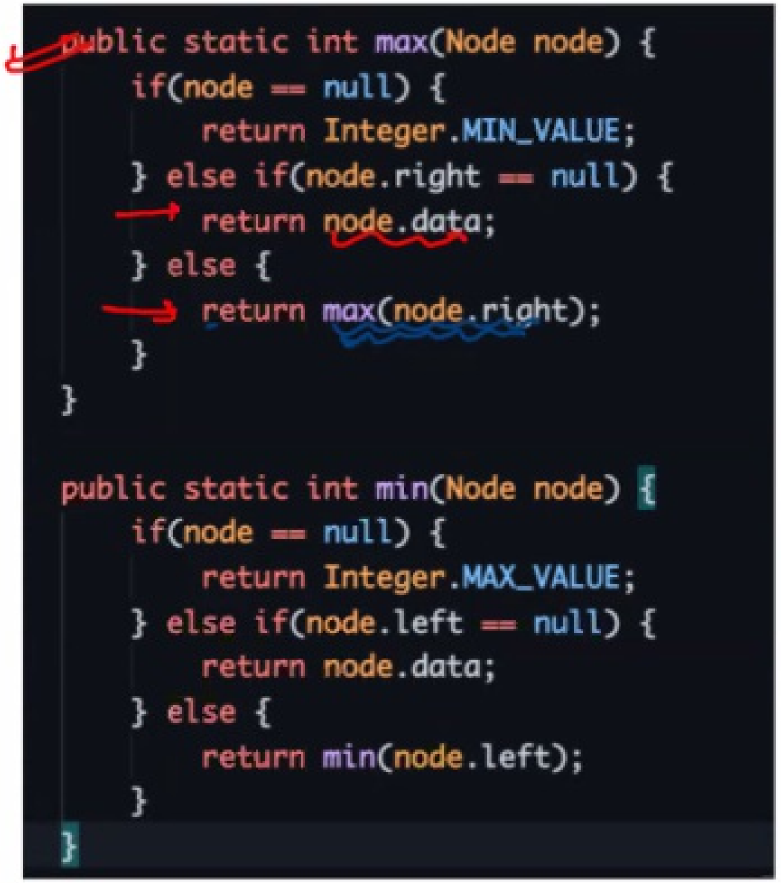


```
public static int max(Node node) {
    if(node == null) {
        return Integer.MIN_VALUE;
    } else if(node.right == null) {
        return node.data;
    } else {
        return max(node.right);
    }
}

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);
    }
}
```

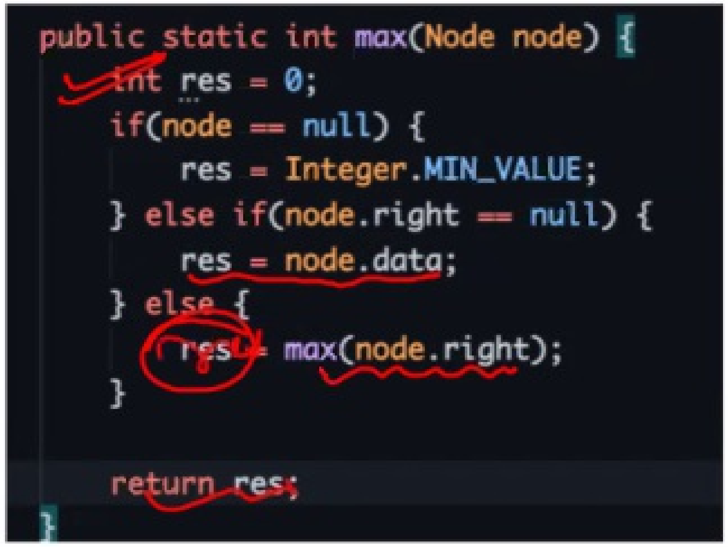


```
public static int sum(Node node) {
    if(node == null) return 0;

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



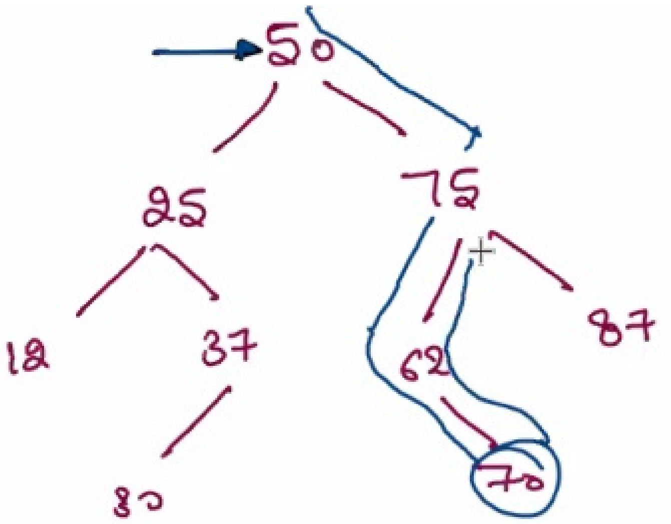


```
public static int max(Node node) {
    if(node == null) {
        return Integer.MIN_VALUE;
    } else if(node.right == null) {
        return node.data;
    } else {
        return max(node.right);
    }
}

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);
    }
}
```



```
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

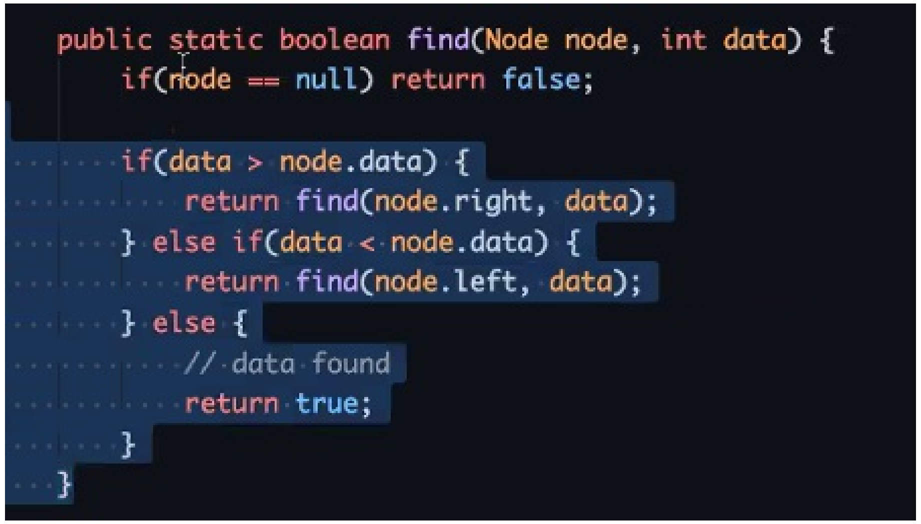


dtf = 70

dtf = 35.

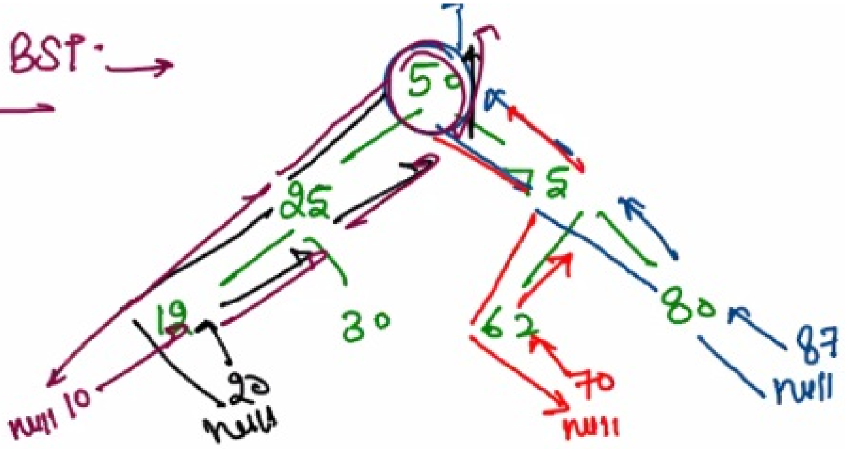
```
if( data > root.data) {
    // Right side.
    return find(root.right, dtf);
} else if (data < root.data) {
    // left side
    return find(root.left, dtf);
} else { // data == root.data
    // data found
    return true;
}
```

[redacted]



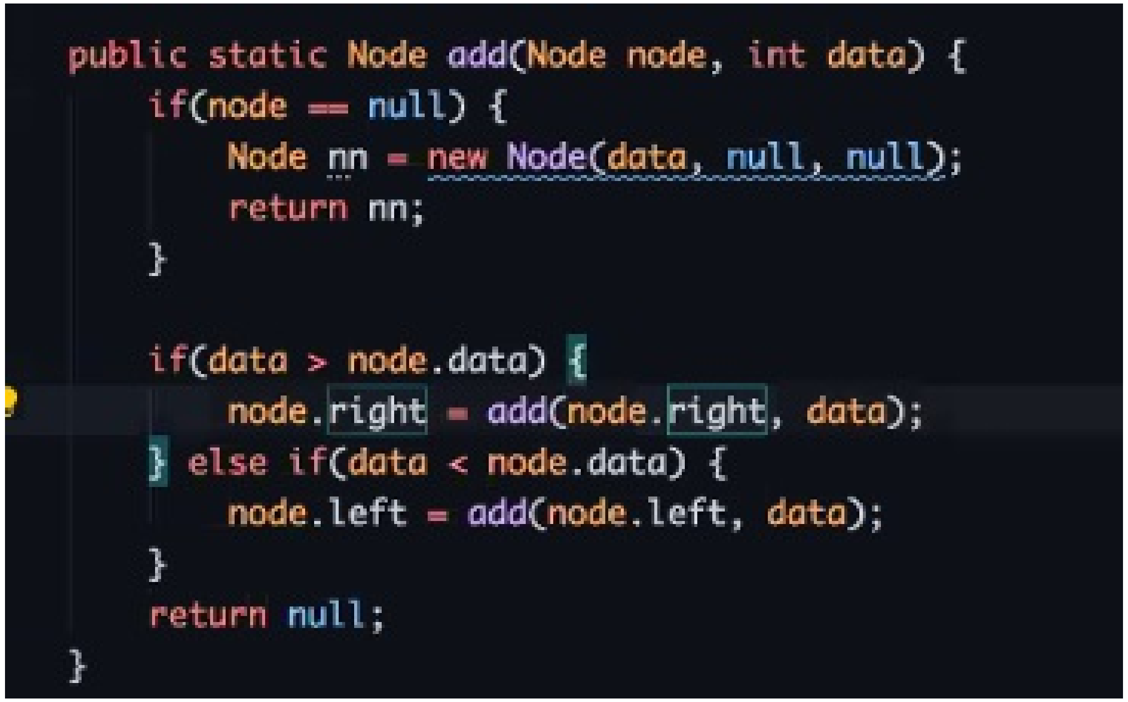
```
public static boolean find(Node node, int data) {
    if(node == null) return false;

    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: →

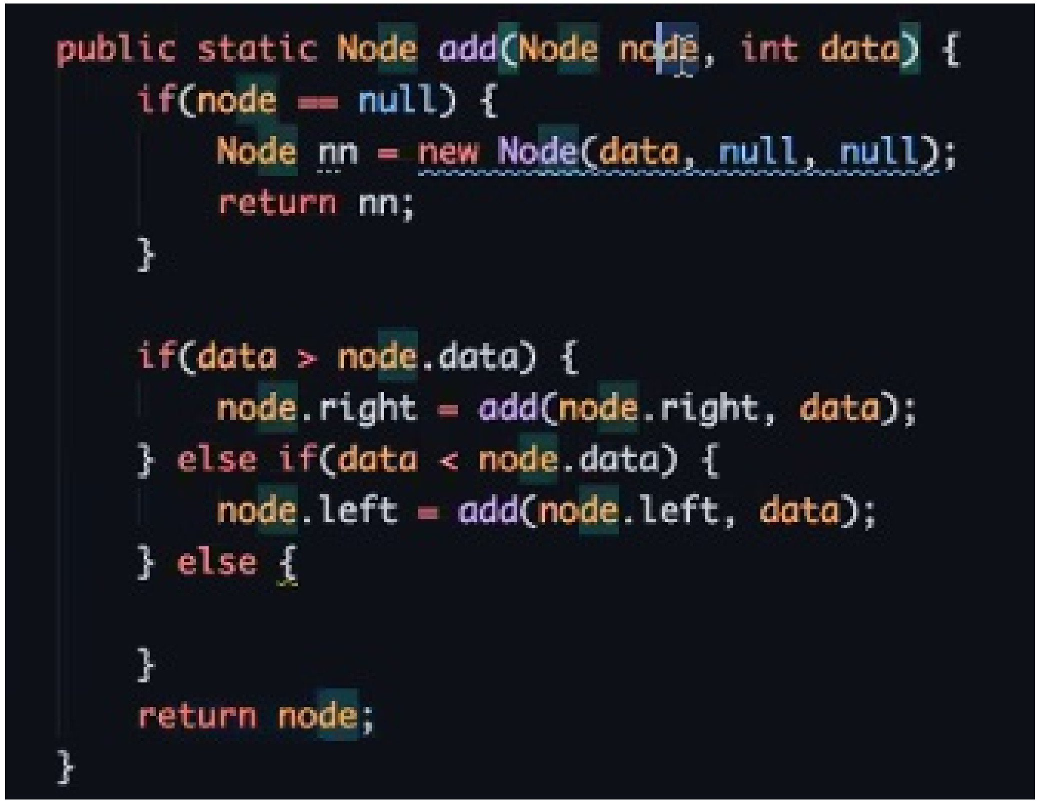


add → 87
add → 70
add → 20
add → 10

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;
}
```