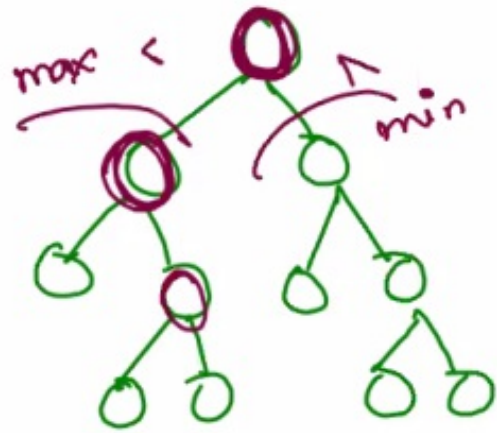


for BST

→ # all nodes in left subtree is smaller than 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.

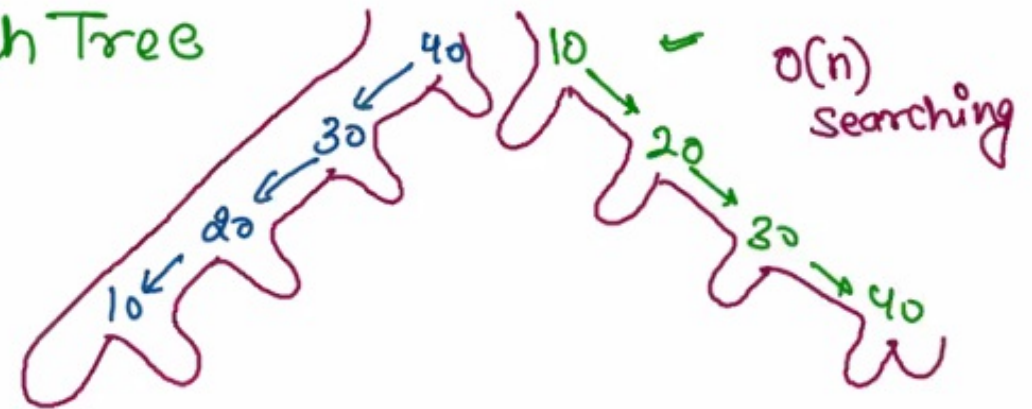
BST → Binary Search Tree
AVL → Balanced BST

Construction

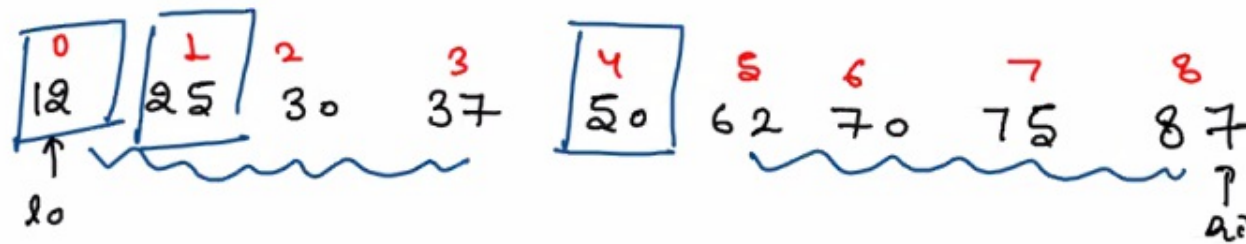
InOrder → Sorted

10 20 30 40

10, 20, 30, 40

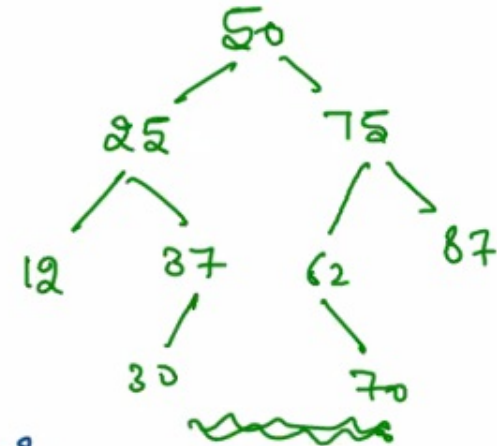
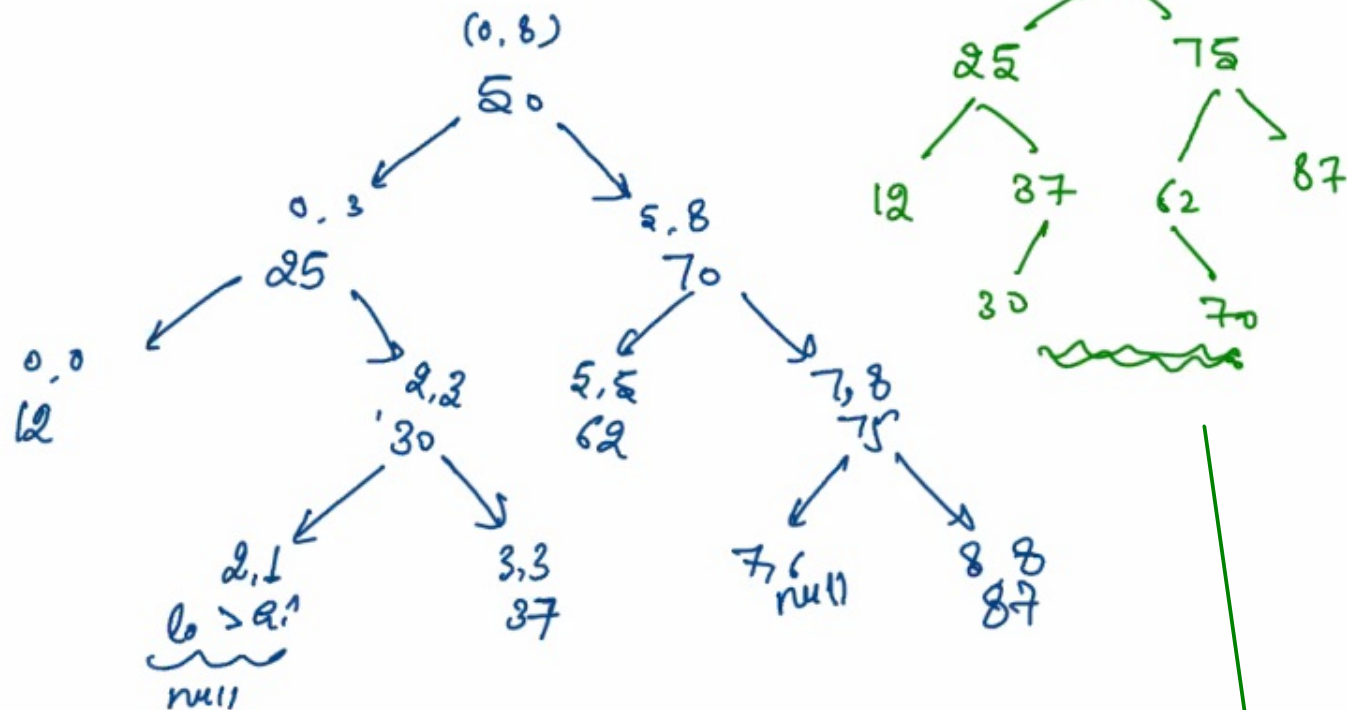
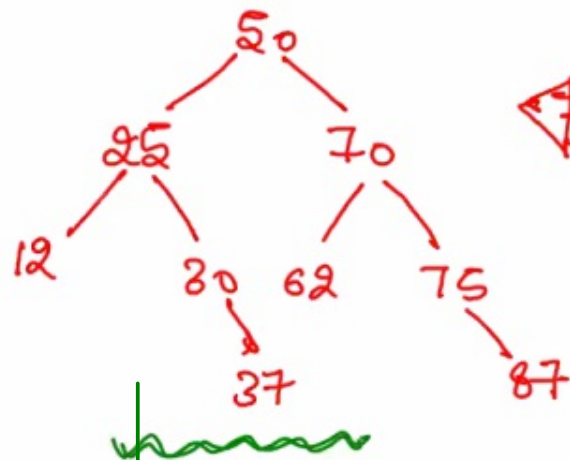


Construction



lo=0
hi=8

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



In Order → 12 25 30 37 50 62 70 75 87

inorder

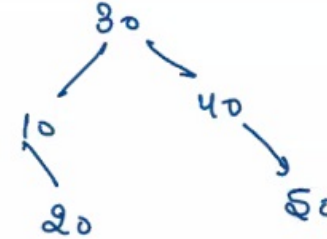
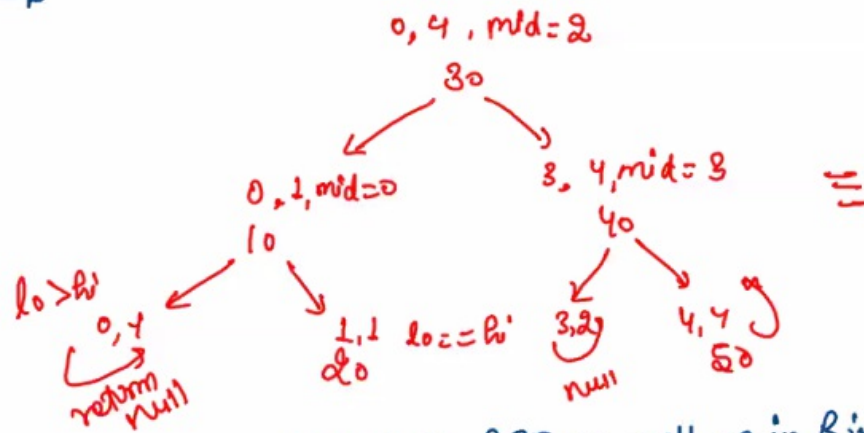
inorder

tree are different but inorder will be same. so that is if inorder is given , you can have multiple BST

data →

0	1	2	3	4
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 \rightarrow Right most Node

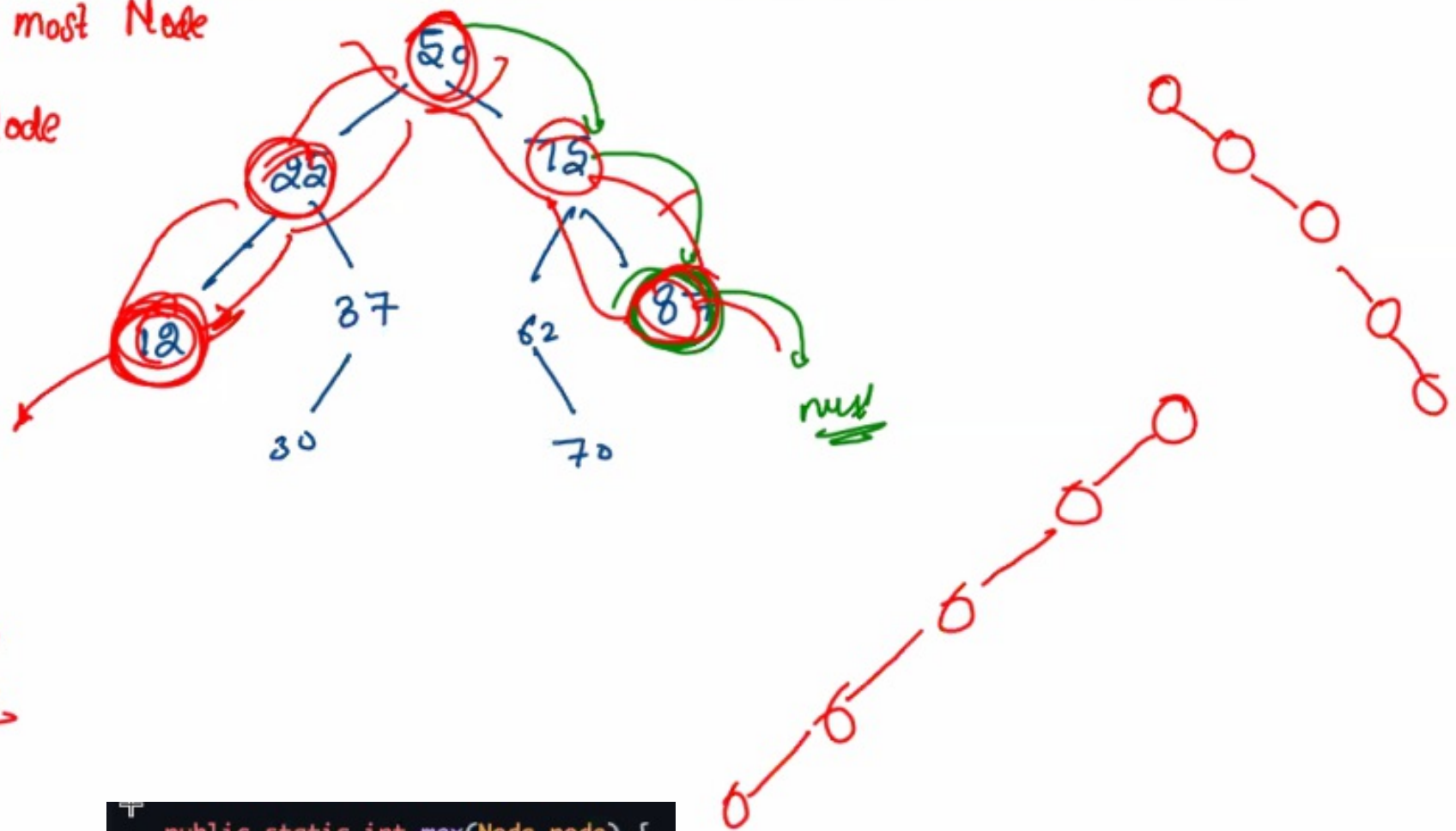
min \rightarrow left most Node

$O(h)$

h is height⁺

AVL \rightarrow $\log n$

height = $\log n$



```
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;

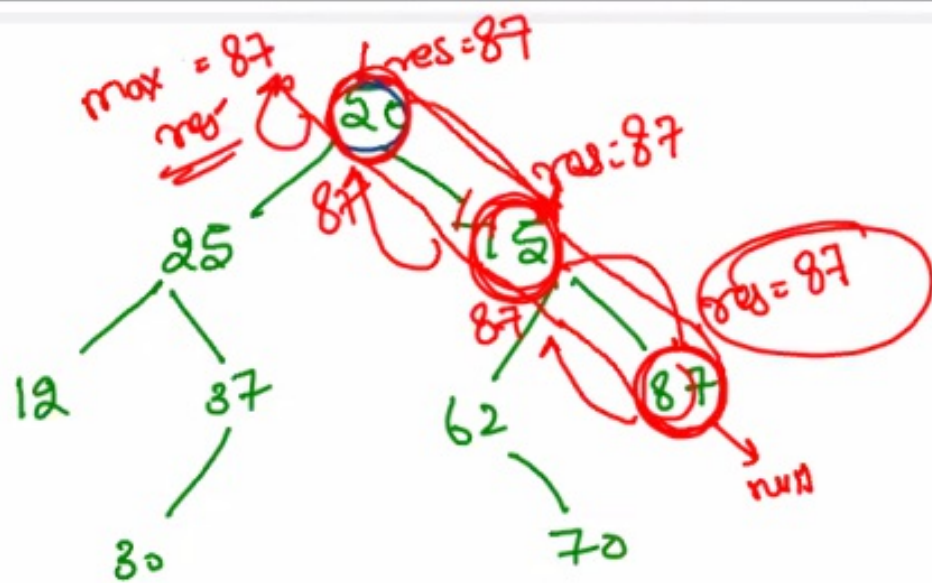
    int lsum = sum(node.left);
    int rsum = sum(node.right);
    return lsum + rsum + node.data;
}
```

```
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);
    }
}

```

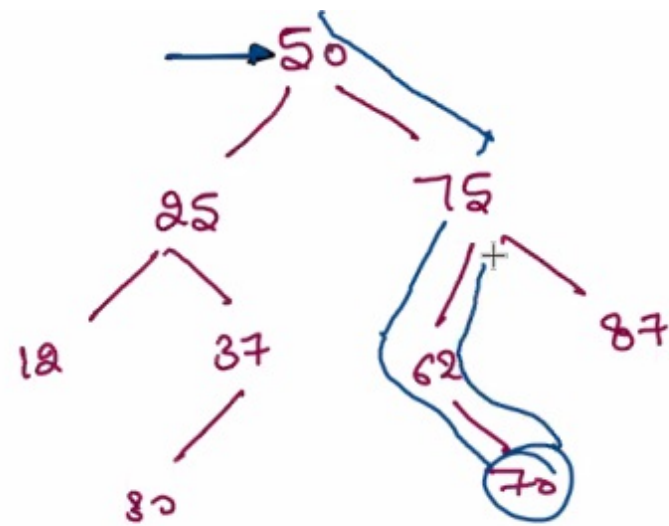
```

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

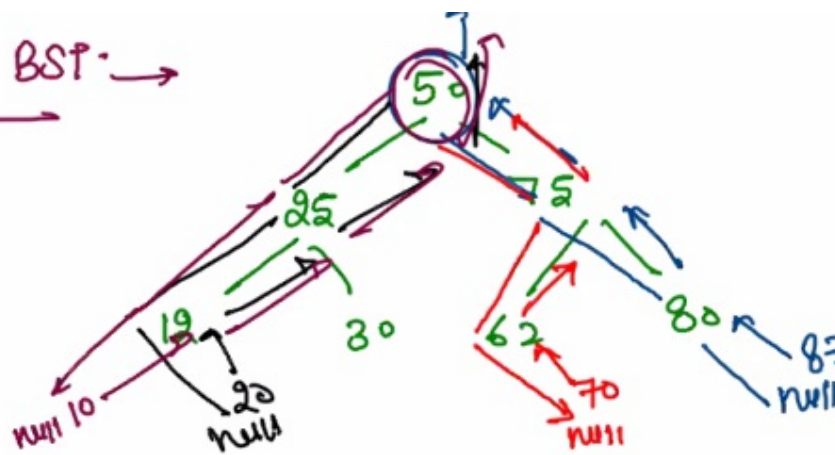


dtf = 70
dtf = 35.

```
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;  
}
```

```
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;
    }
}
```

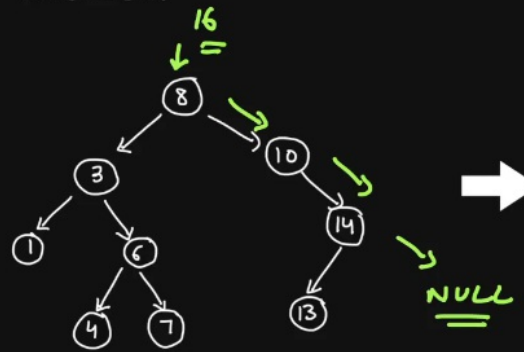



Closest in BST

↑

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 $\rightarrow 8$ & 14
 $16 - 10 = 6$
 $16 - 14 = 4$

Output : 14

50 / 2:53

Target Value (16)

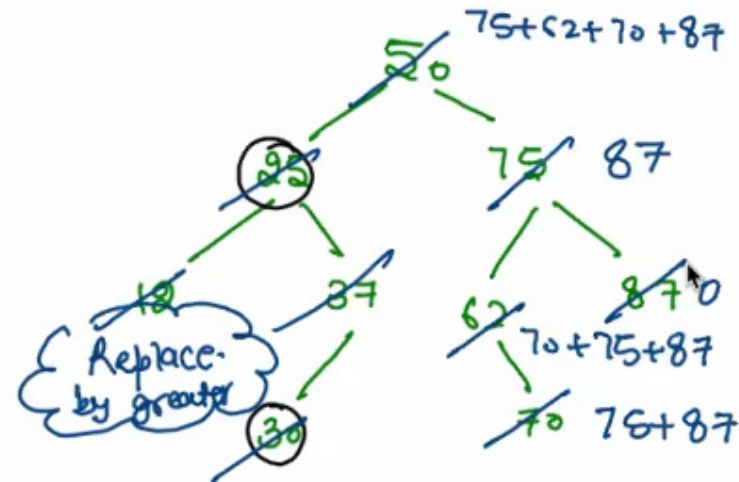
```

33 public int closestDiff(Node root, int target){
34
35     int closest = 0;
36     int diff = Integer.MAX_VALUE;
37
38     Node temp = root;
39
40     while(temp != null){
41         int current_diff = Math.abs(temp.value - target);
42
43         if(current_diff == 0){
44             return temp.value;
45         }
46
47         if(current_diff < diff){
48             diff = current_diff;
49             closest = temp.value;
50         }
51
52         if(temp.value < target){
53
54             temp = temp.right;
55
56         } else {
57
58             temp = temp.left;
59
60         }
61     }
62     return closest;
63 }

```


Replace Sum of its larger value

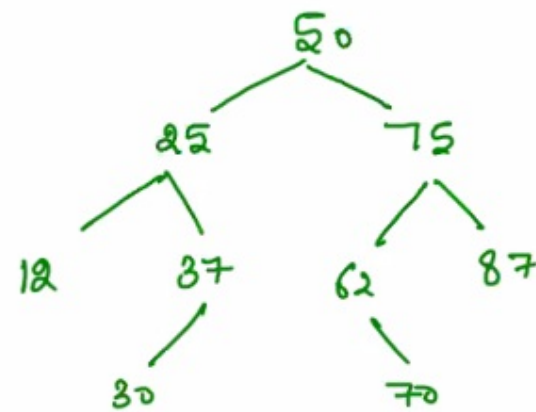
→ node data Replace by
sum of its larger
value.



$$\frac{25}{X} \rightarrow \text{sum of its greater value} = \underbrace{80 + 37 + 50 + 62 + 70 + 75 + 87}_X$$

Time complexity $\rightarrow O(n)$

Space : Recursive $O(\text{height})$



+

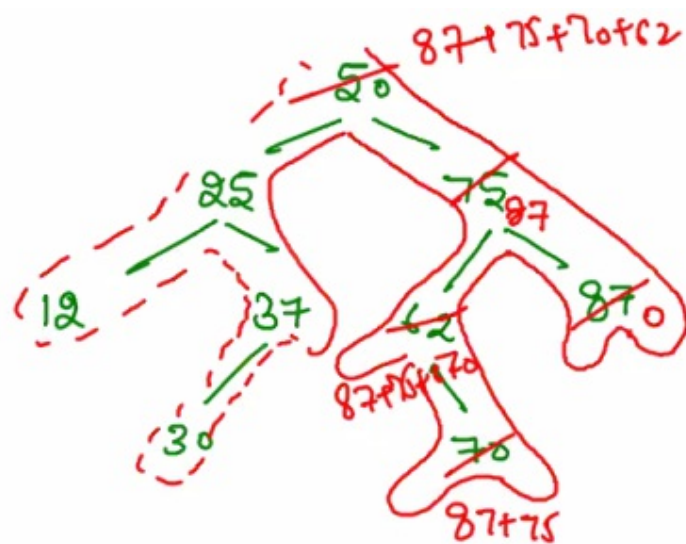
InOrder → Sorted Number

Reverse InOrder → Decreasing order.

Sum
less
than
value

	10	20	30	40	50
Sum	0	10	30	60	100
less than value	sum = 0	10	20	30	100
	140	120	90	50	0
Sum greater than value	sum = 140	50	90	120	150

Tree:



InOrder \rightarrow Increasing order

Reverse InOrder \rightarrow Decreasing order

$$\text{sum} = 0 + 87 + 75 + 70 + 62 + 50$$

Example:

arr \rightarrow 10 20 30 40 50
 new values \Rightarrow 0 10 30 60 100
 replace value by sum less than
 data $\equiv \text{arr}[i]$
 sum \Rightarrow 0 10 30 60 100 150

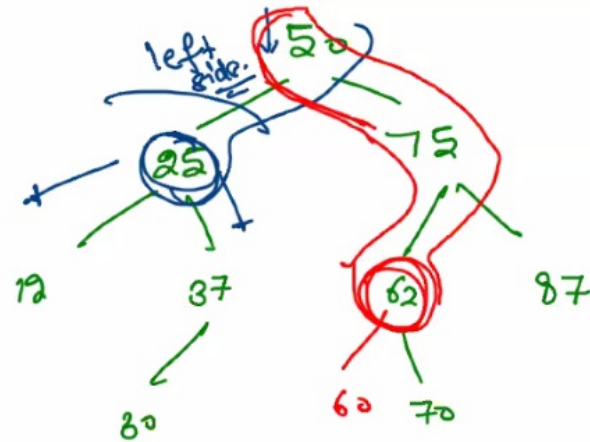
replace value by values = 140 120 90 50 0
 sum greater than
 value $\equiv \text{arr}[i]$
 sum = 0 50 90 120 140 150

Step 1: Traverse - Reverse Inorder
 1. get data.
 2. Replace value by sum
 3. provide sum part of data on sum.

3-4

```
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



25
62

12, 87

60-70

Case-I $\iff d1 > \text{node.data} \quad d2 > \text{node.data}$
 \longrightarrow right side,

Case-II $\iff d1 < \text{node.data} \quad d2 < \text{node.data}$
 \longrightarrow left side,

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

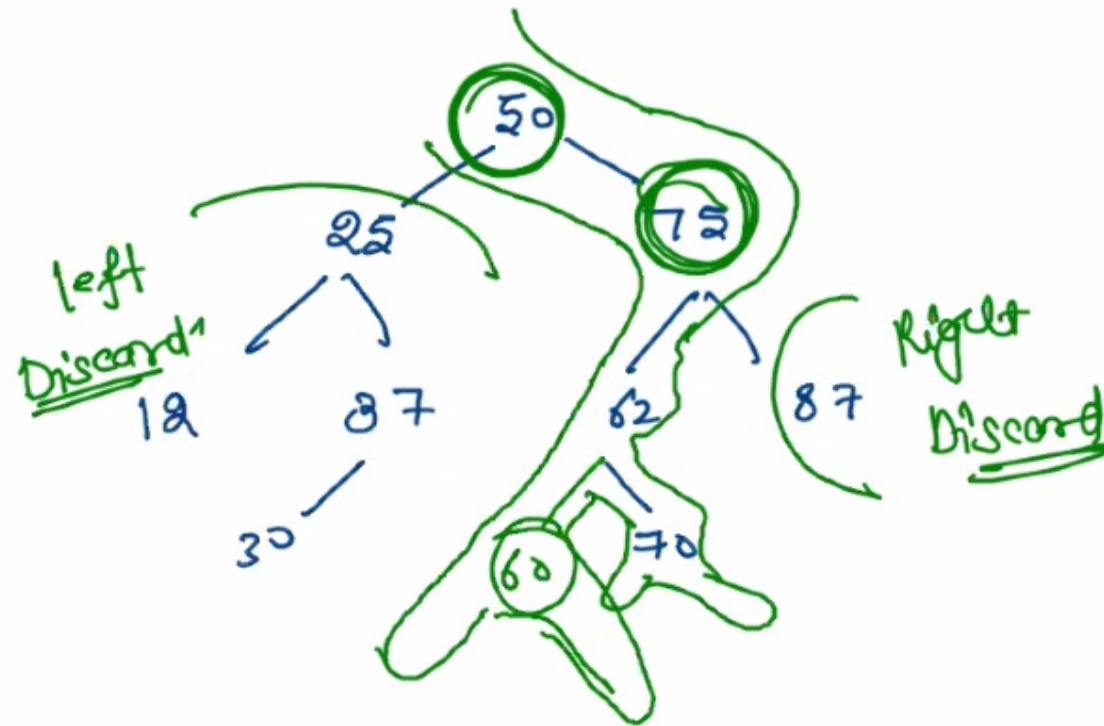
```

5
7 public static int lca(Node node, int d1, int d2) {
3     if(d1 > node.data && d2 > node.data) { // right side
3         return lca(node.right, d1, d2);
0     } else if(d1 < node.data && d2 < node.data) { // left side
1         return lca(node.left, d1, d2);
2     } else { // answer
3         return node.data;
4     }
5 }

```


print in Range

60 73



~~~~~  
60 62 70<sup>+</sup>  
~~~~~

```
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

Method 1

Time.

nh
traversal + find

Space.

h (Recursive)

Method 2

Ind:
left pointer
right pointer

n

n
Arraylist fill \rightarrow 9n Area
Sorted arraylist

Method 3

n
better

h
better

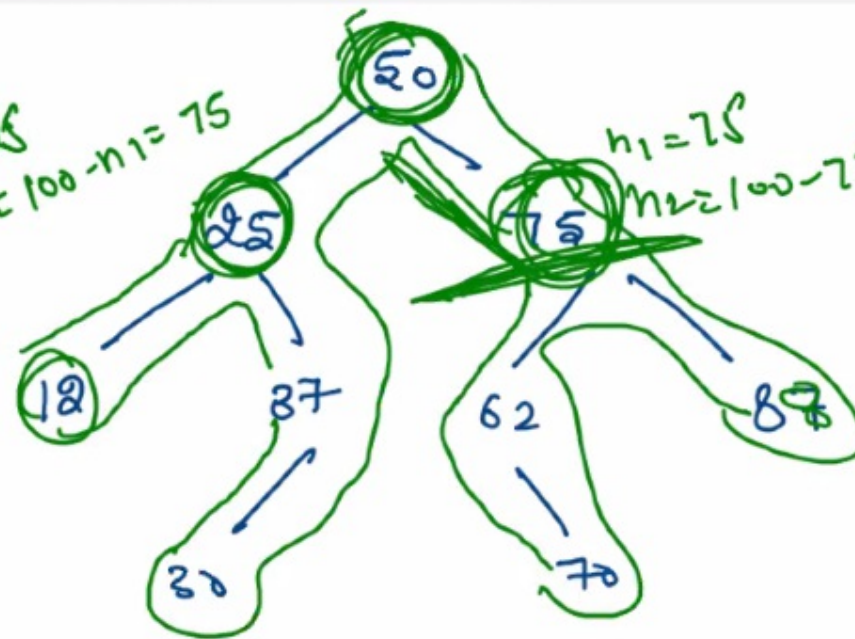
25 75
12 88
30 70

$n_1 = 12$
 $n_2 = 88$

12 88
25 70
+ 30 70

$n_1 = 25$
 $n_2 = 100 - n_1 = 75$

$n_1 = 75$
 $n_2 = 100 - 75 = 25$



Method 1

+

nh
traversal + find

h (Recursive)

// method 1, time : $O(nh)$, space : $O(h)$, $h \rightarrow$ 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

Ans:

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--;
        }
    }
}
```


method 3

n

h

target = 90

10 - 80

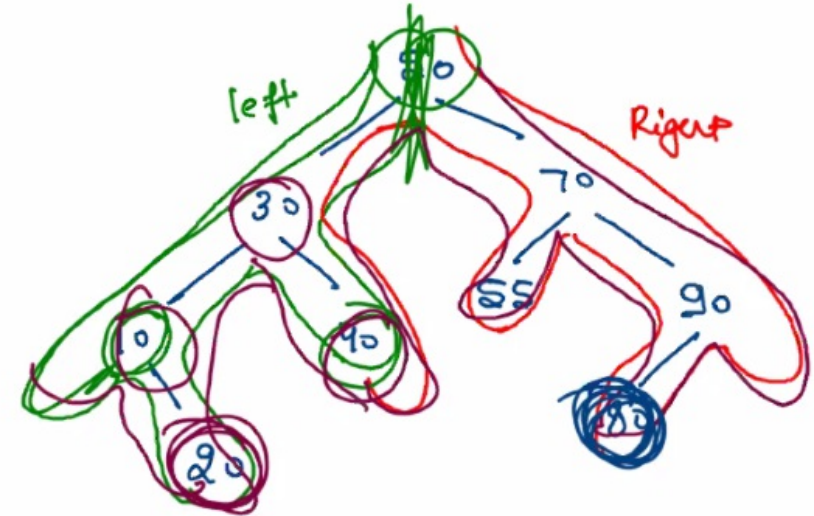
20 - 70

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
↳ left child push in stack state++
state = 1
↳ right child push state++ } done
state = 2
↳ pop

left stack - normal iterative

state = 0
↳ right child state++
state = 1
↳ left child state++ } done
state = 2
↳ pop



reverse right stack - iterative

```
// 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);
        }
    }
}
```

```

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;
}

```

```

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;
}

```

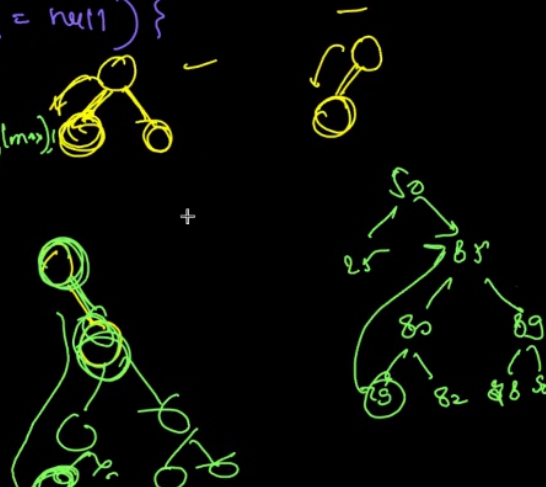
data found

```
if( node.left == null && node.right == null ) {  
    return null; leaf node  
}
```

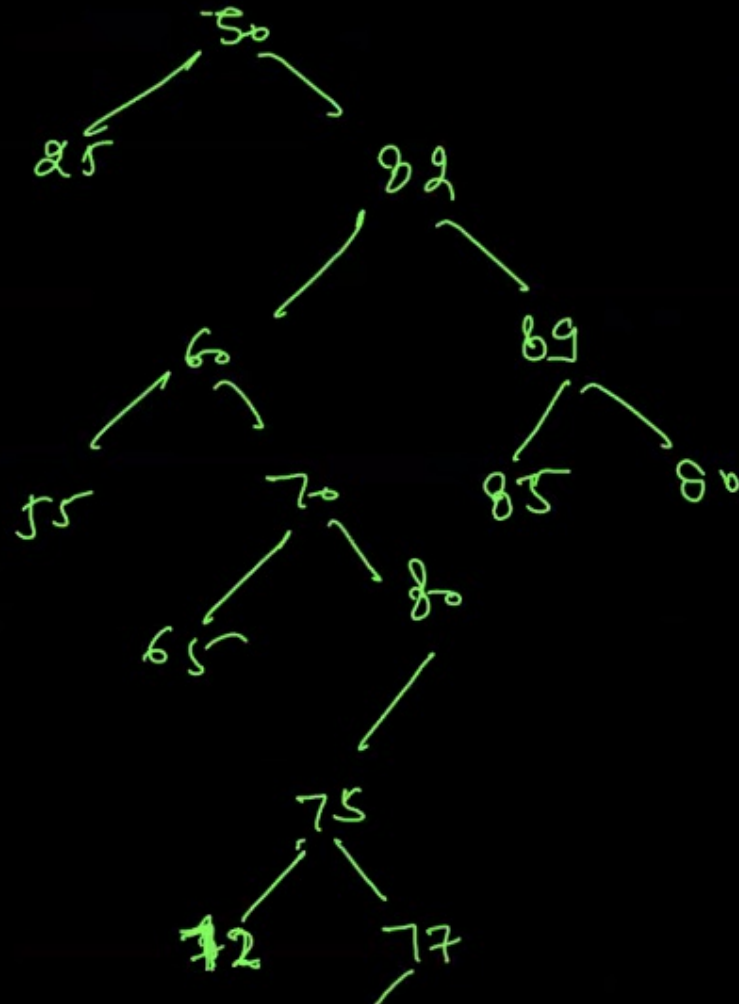
```
} else if( node.left != null ) {  
    lmax = max(node.left);  
    node.data = lmax;  
    node.left = remove( node.left, lmax );  
}
```

```
} else {
```

```
    return node.right;  
}
```



```
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 + RUTH