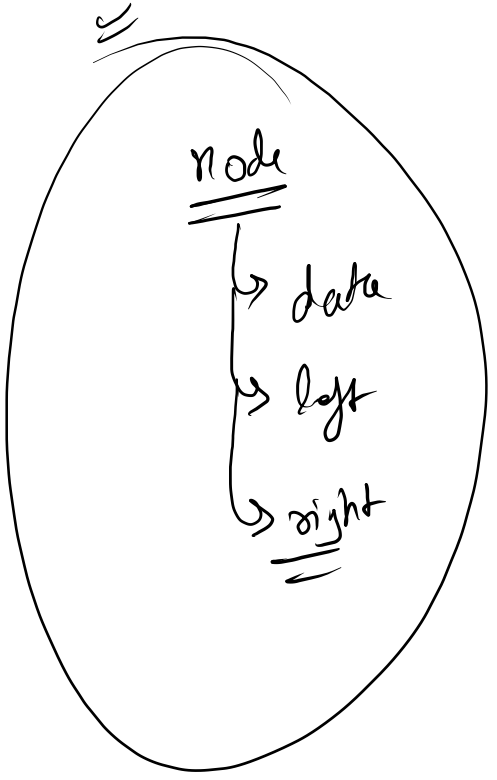
B.S.T → structure similar: B.T. + B.S.T. Property

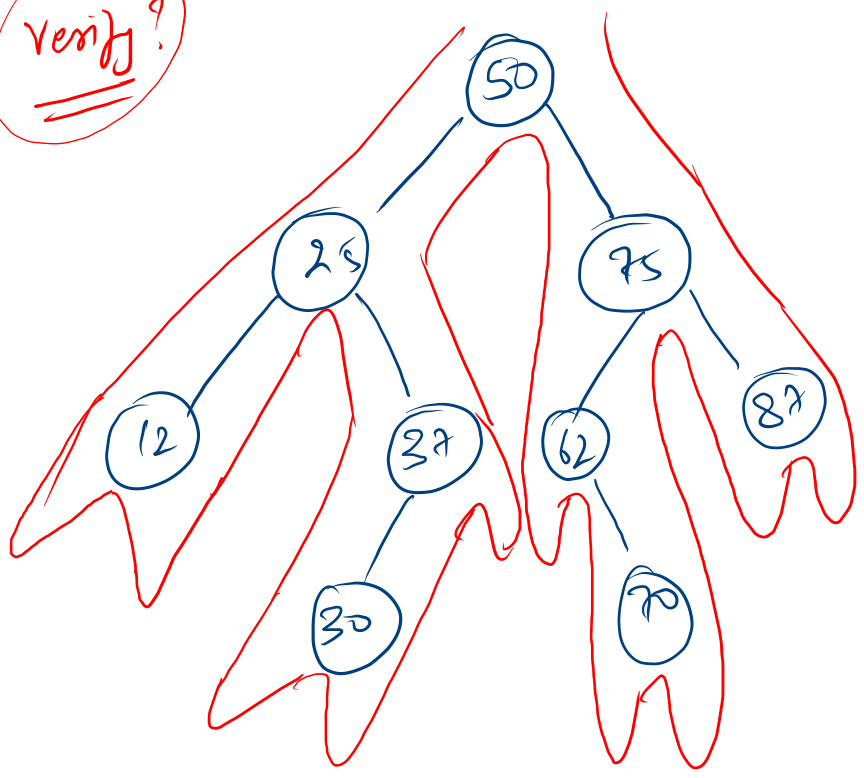
all (left subtree) < node.data < (right subtree) all, ∀ nodes

	B.S.T.	B.T.
Display	O(n)	O(n)
Sum	O(n)	O(n)
Find	optimization	O(n)

Algo
- structure oriented
- data =



Verify?



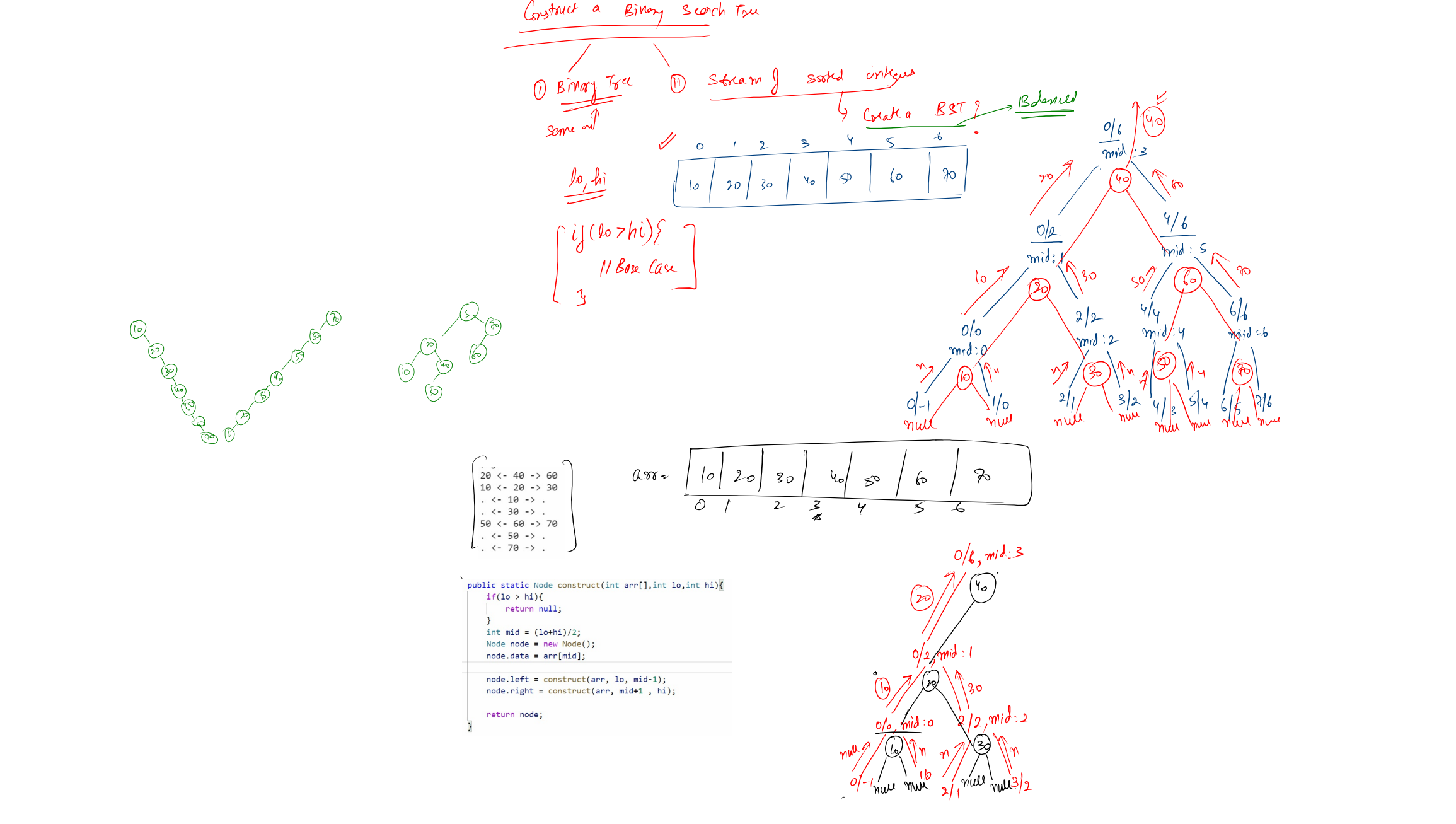
* Property → BST → inorder → increasing order

↓

LNR

12 25 30 37 50 62 70 75 87

→



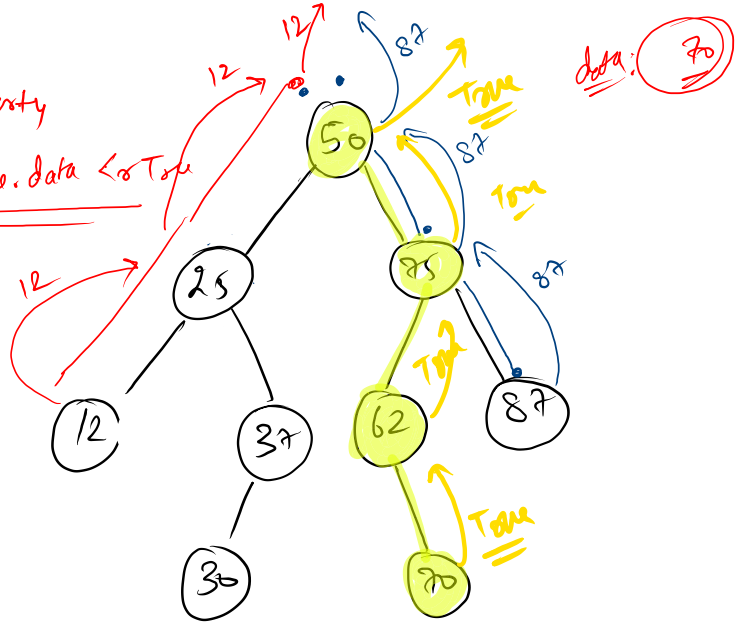

```java
public static Node construct(int arr[],int lo,int hi){
    if(lo > hi){
        return null;
    }
    int mid = (lo+hi)/2;
    Node node = new Node();
    node.data = arr[mid];

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

    return node;
}
```

```
19
50 25 12 n n 37 30 n n n 75 62 n 70 n n 87 n n
70
```

```java
public static int size(Node node) {
  // write your code here
}

public static int sum(Node node) {
  // write your code here
}

public static int max(Node node) {
  // write your code here
}

public static int min(Node node) {
  // write your code here
}

public static boolean find(Node node, int data){
  // write your code here
}
```

BST → BST Property

left Tree < node.data < right Tree

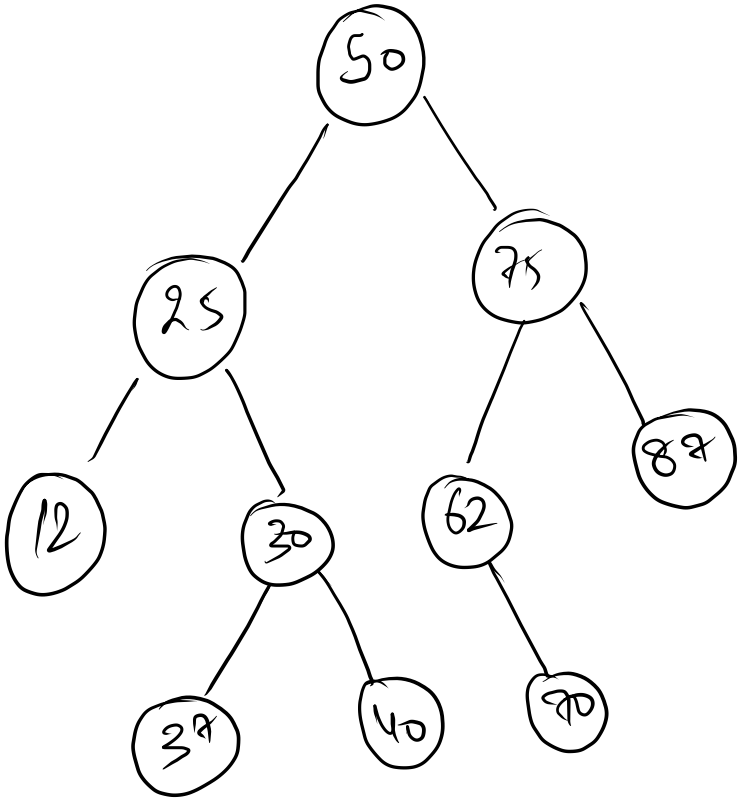


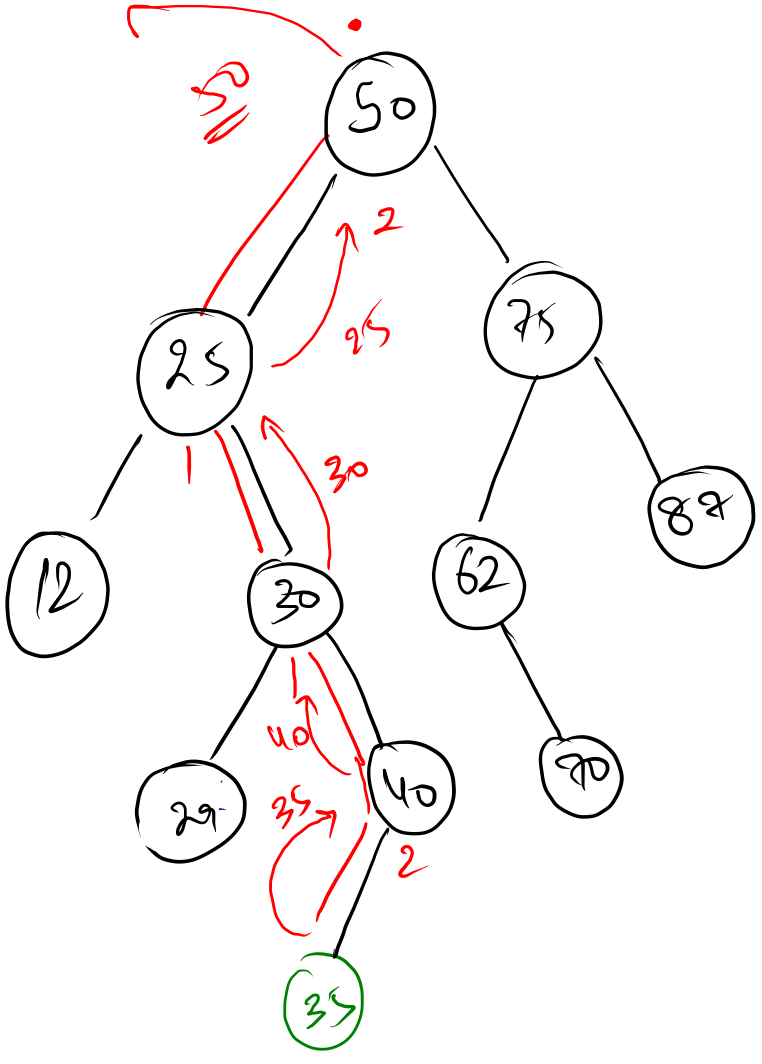

```java
public static int max(Node node) {
  if (node.right == null) {
    return node.data;
  } else {
    return max(node.right);
  }
}

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

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

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

Add Node To Binary Search Tree

Data : 35

BST should be satisfied





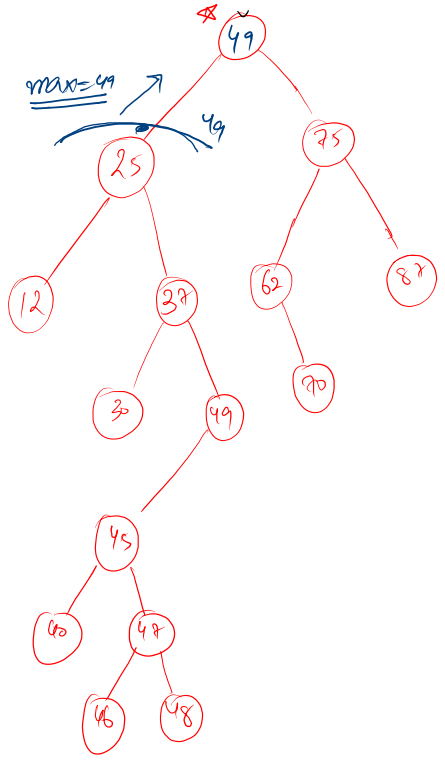
Data : 35

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

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

REMOVE NODE FROM BST

Remove 50

12 25 30 37 40 45 46 47 48 49 50 62 70 75 87




```
}else{ // both child exists
    int max = max(node.left);//49
    node.data = max;
    remove(node.left,max);
    return node;
}
```

Remove node
- → doesn't Exist
- → Leaf Node
- → single child
- → both child

Remove 50

```
public static Node remove(Node node, int data) {
    if(node == null){
        return null;
    }
    if(data < node.data){
        node.left = remove(node.left,data);
    }else if(data > node.data){
        node.right = remove(node.right,data);
    }else{
        if(node.left == null && node.right == null){ // leaf node
            return null;
        }else if(node.left == null){ // single child : right child exists
            Node rep = node.right;
            node.right = null;
            return rep;
        }else if(node.right == null){ // single child : left child exists
            Node rep = node.left;
            node.left = null;
            return rep;
        }else{ // both child exists

        }
    }
```

15

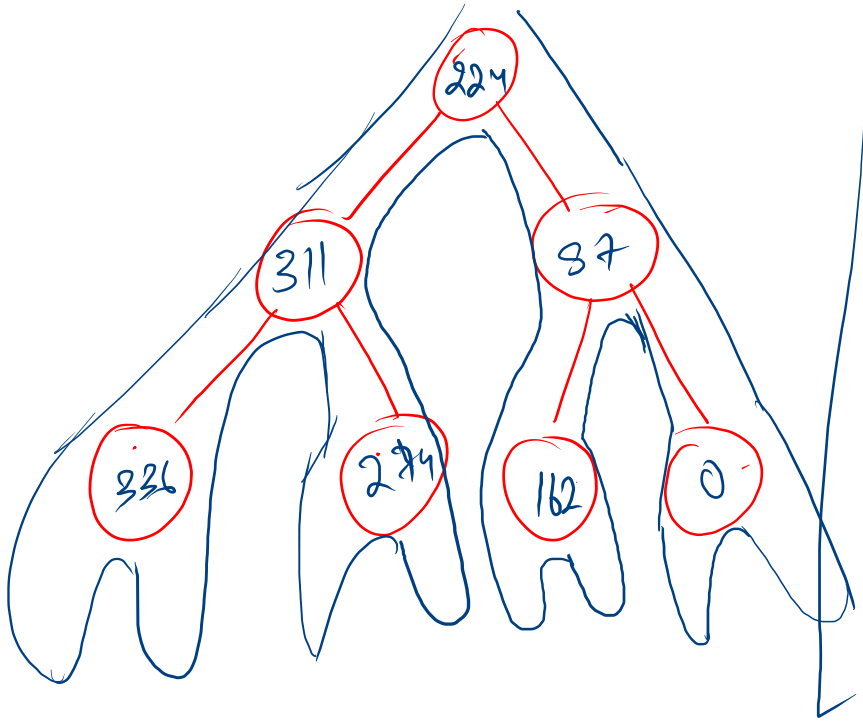
50 25 12 n n 37 n n 75 62 n n 87 n n

Reverse Inorder

RNL

Sum = 0 + 87 + 75 + 62 + 50 + 37 + 25

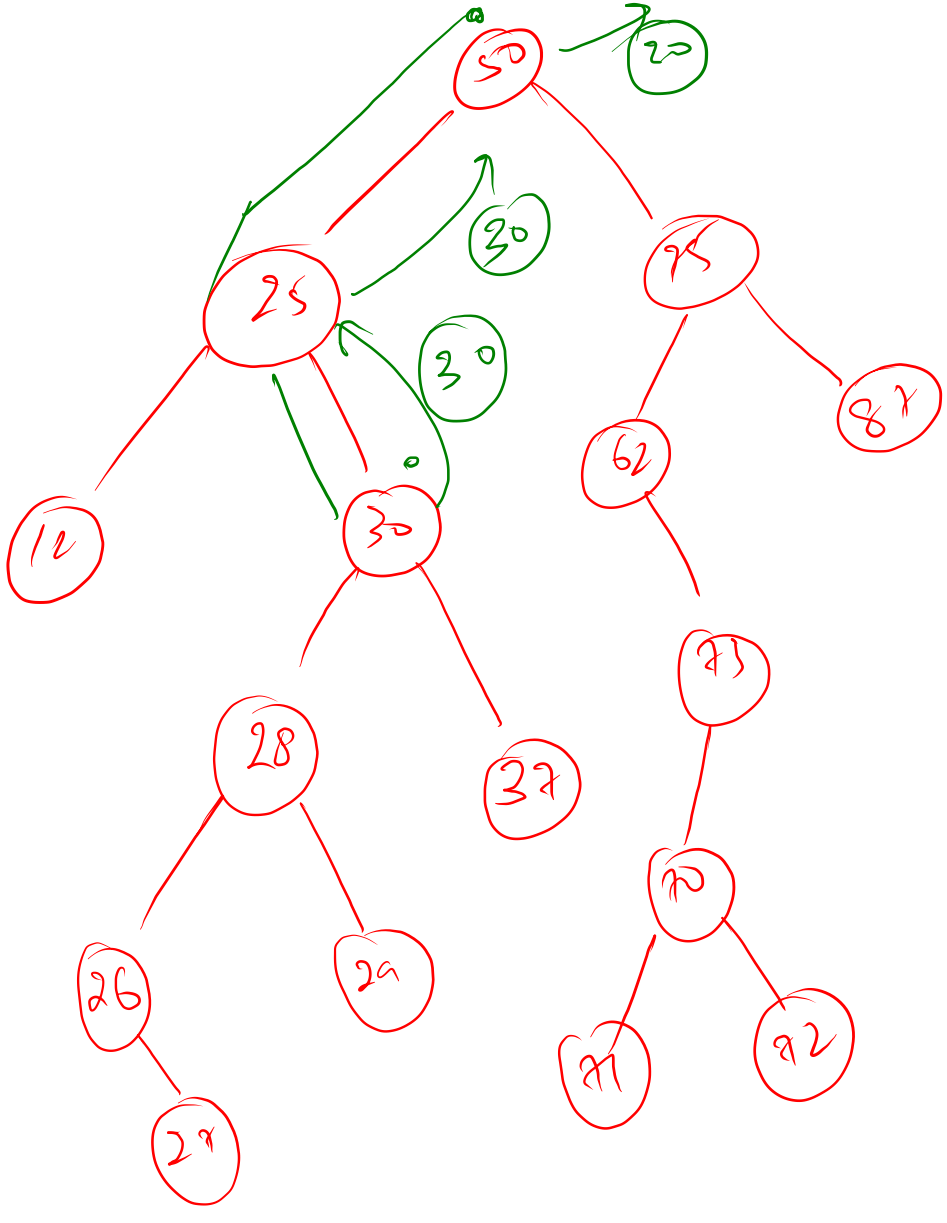
val: 25




```java
static int sum = 0;
public static void rwsol(Node node){
  if(node == null){
      return;
  }
  rwsol(node.right);

  // reverse inorder
  int val = node.data;
  node.data = sum;
  sum += val;

  rwsol(node.left);
}
```

LCA

H.W.



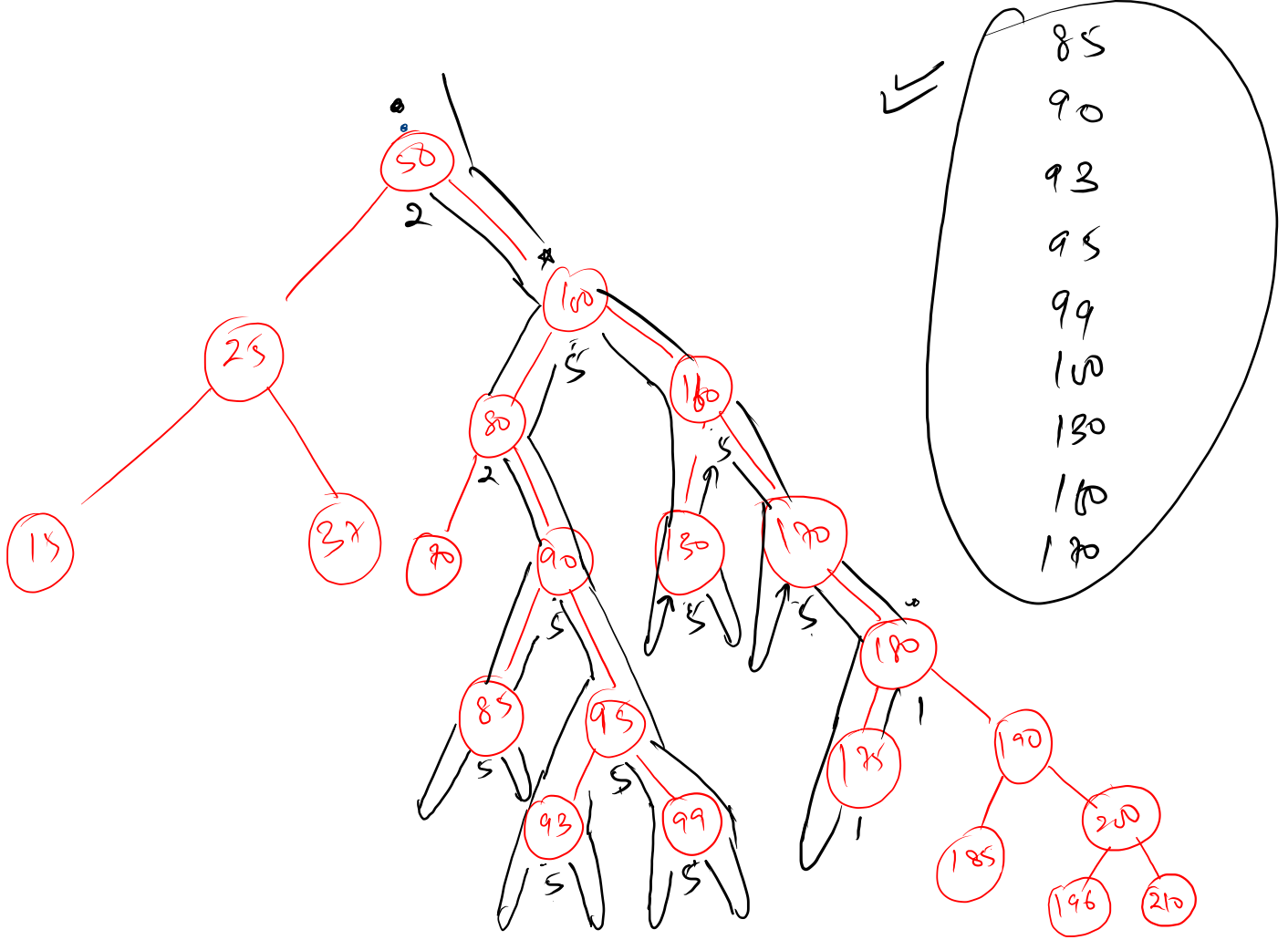
Mint

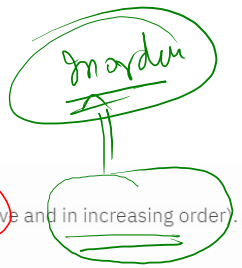
2^{d_1} , 3^{d_2}

```
if( d1 < n.d && d2 < n.d){
    LCA(n.lgt);
}
else if(d1 > n.d && d2 > n.d){
    LCA(n.right)
}else{
    return node;
}
```

```
public static void pir(Node node, int d1, int d2) {
  if(node == null){
      return;
  }
  if(d1 < node.data && d2 < node.data){
      pir(node.left,d1,d2);
  }else if(d1 > node.data && d2 > node.data){
      pir(node.right,d1,d2);
  }else{
      pir(node.left,d1,d2);
      System.out.println(node.data);
      pir(node.right,d1,d2);
  }
}
```

d1 : 85 d2 : 180





1. You are given a partially written BST class.
2. You are required to complete the body of pir function. "pir" function is expected to print all nodes between d1 and d2 (inclusive and in increasing order).
3. Input and Output is managed for you.

21
50 25 12 n n 37 30 n n n 75 62 60 n n 70 n n 87 n n
100

Target: 100

25 75

val + C.val = Tar

✓ ✗ ✗

✓ ✓ ✓

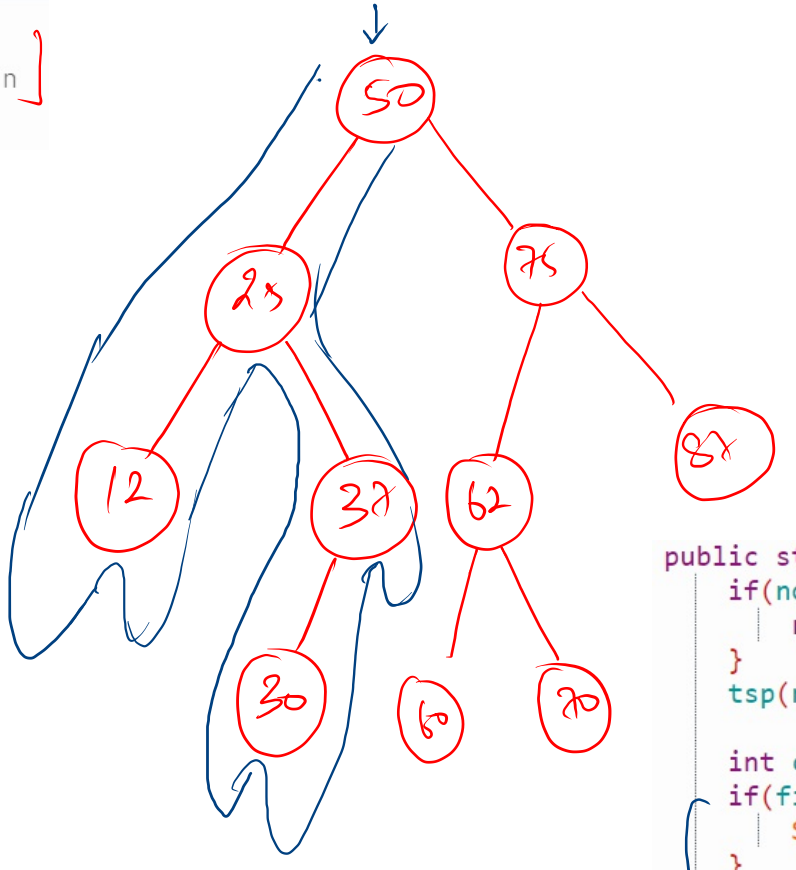
C.val ⇒ tar − n.data

node.data

Cval

Cval > node.data

- 25 75
- 30 70
- 50 50 ✗
- 70 30 ✗
- 75 25 ✗




```
public static void tsp(Node node,int target,Node root){
    if(node == null){
        return;
    }
    tsp(node.left,target,root);

    int cval = target-node.data;
    if(find(root,cval)){
        System.out.println(node.data +" "+cval);
    }

    tsp(node.right,target,root);
}
```