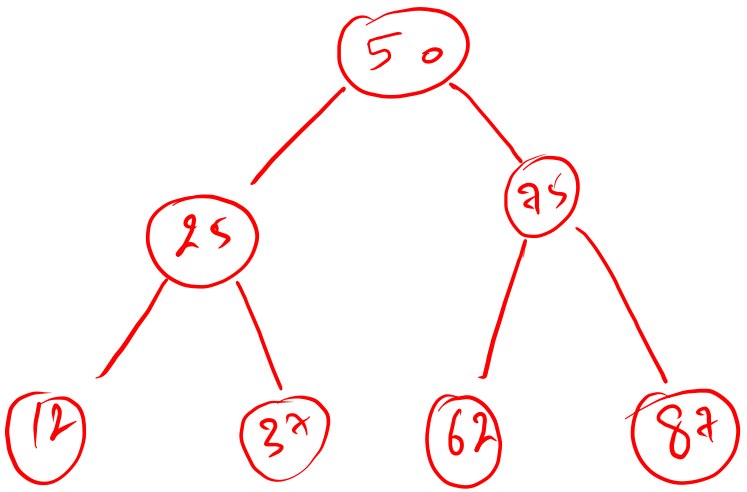
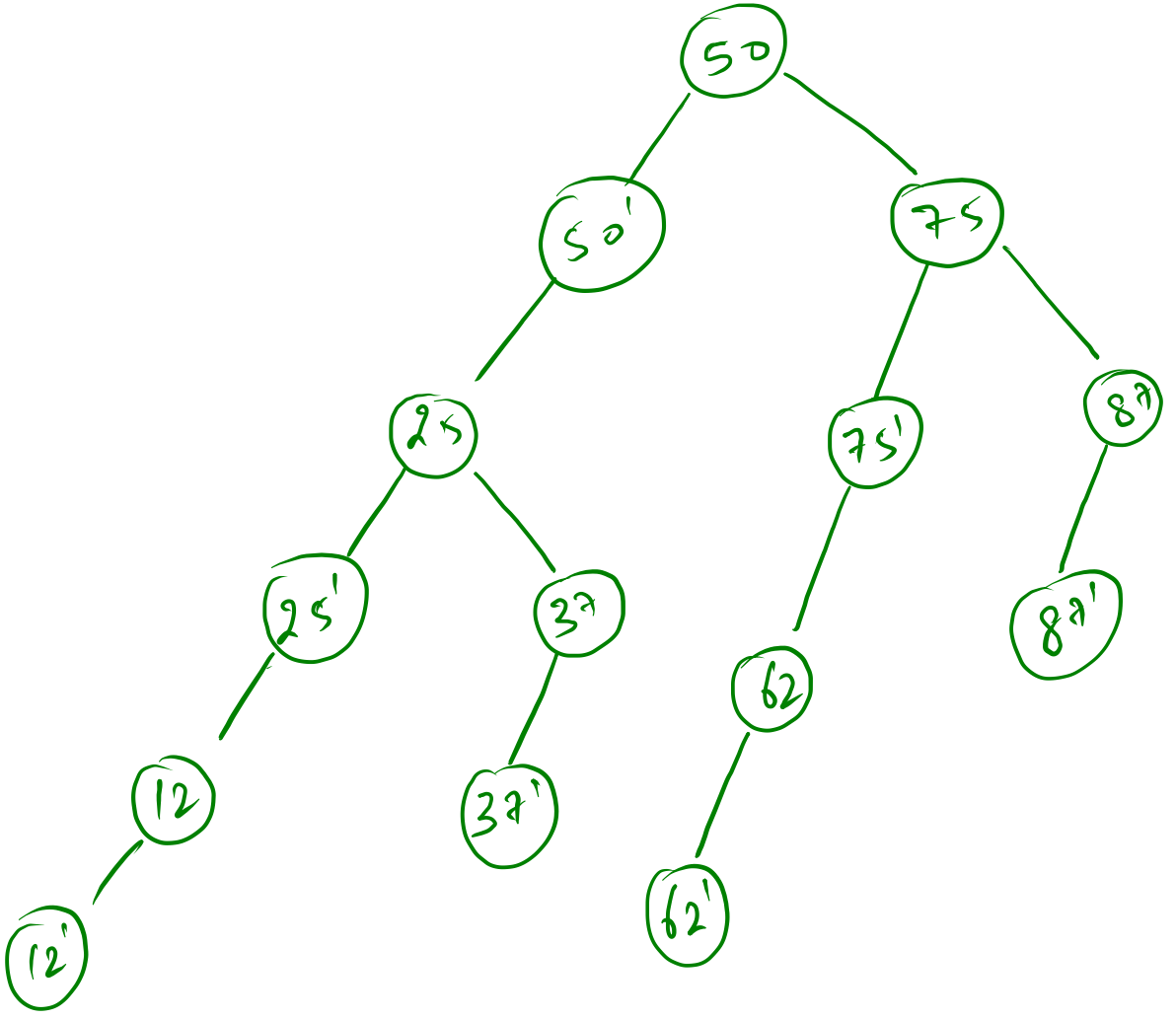
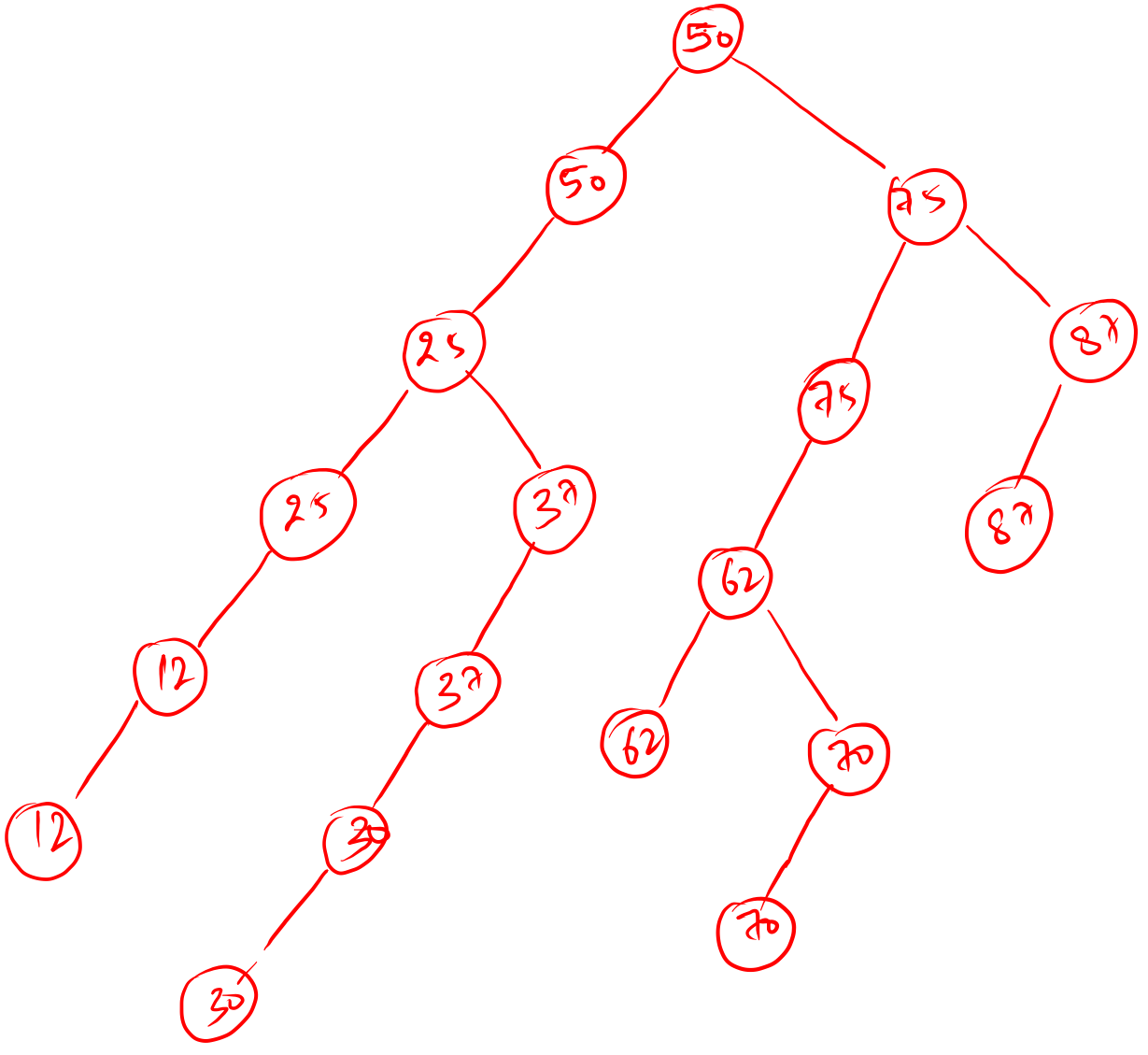
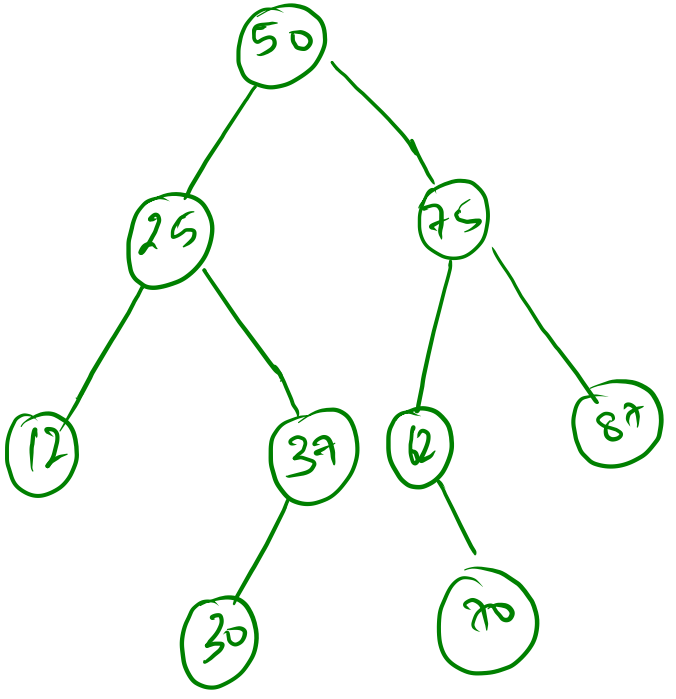
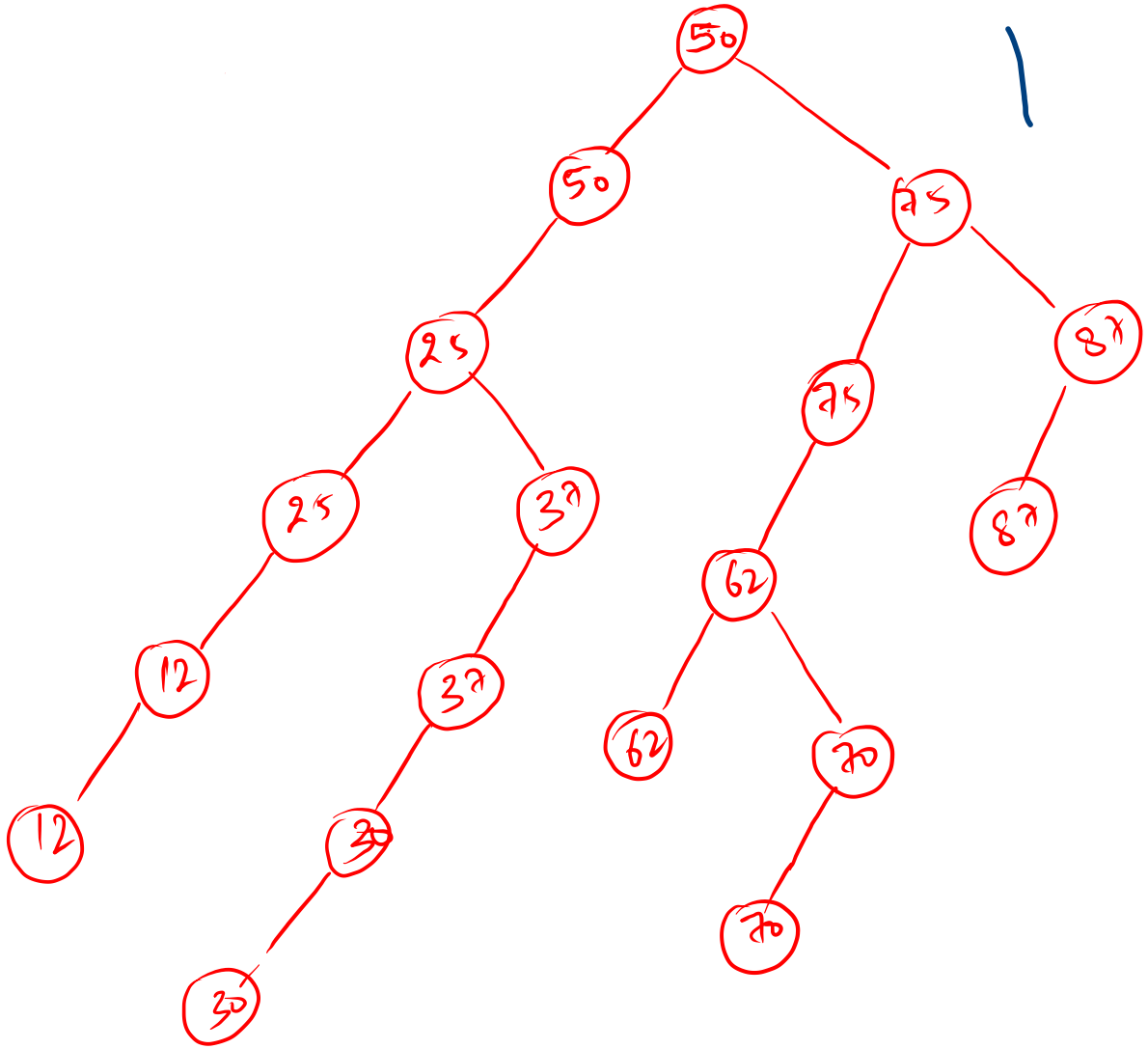
- Transform To Left-cloned Tree
- Transform To Normal From Left-cloned Tree
- Print Single Child Nodes
- Remove Leaves In Binary Tree
- Diameter Of A Binary Tree
- Tilt Of Binary Tree
- Is A Binary Search Tree
- Is Balanced Tree
- Largest Bst Subtree


50
25
75
12
37
62
87


50
50'
75
25
75'
87
25'
37
62
87'
12
37'
62'
12'

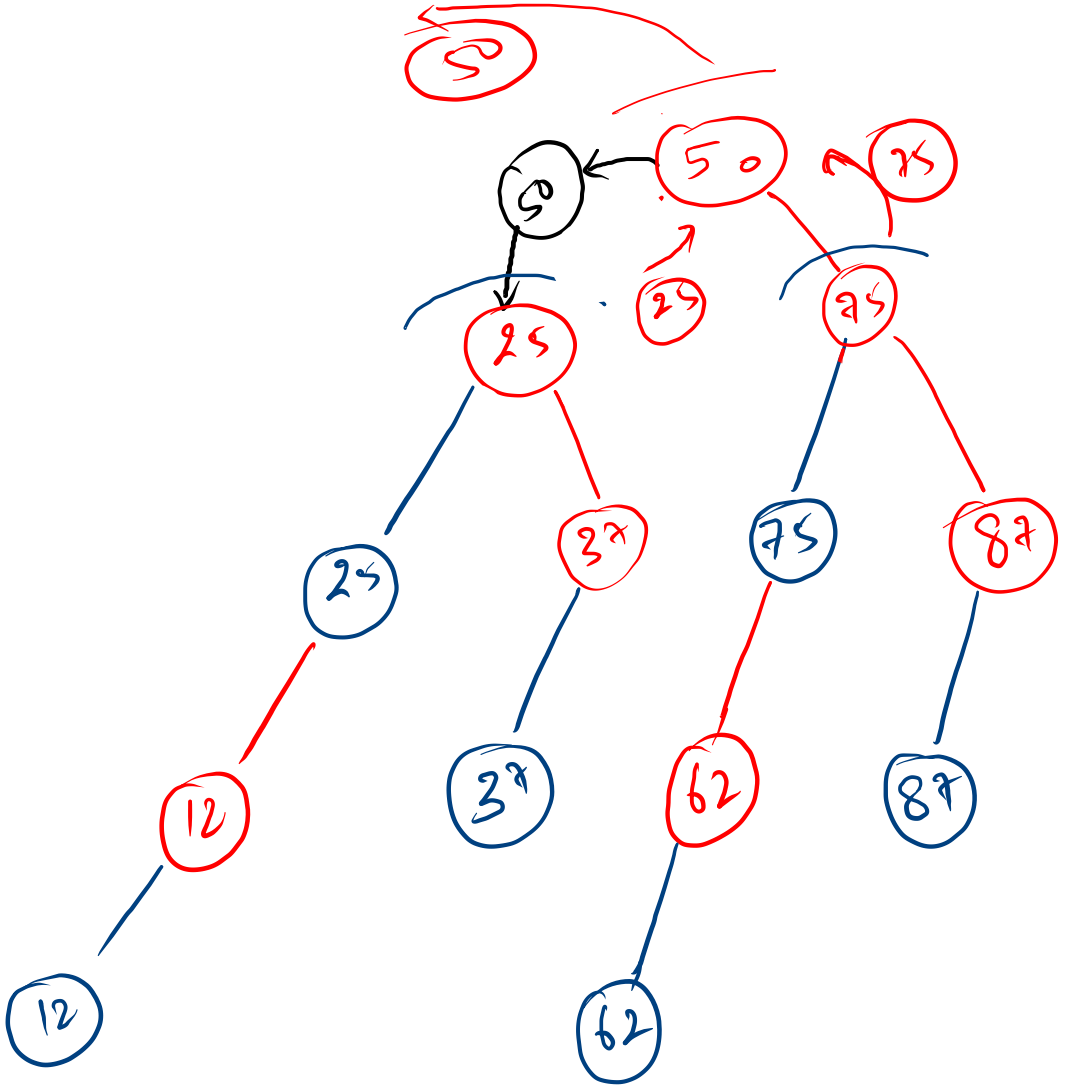

50
50
75
25
75
87
25
37
62
87
12
37
62
70
12
30
70

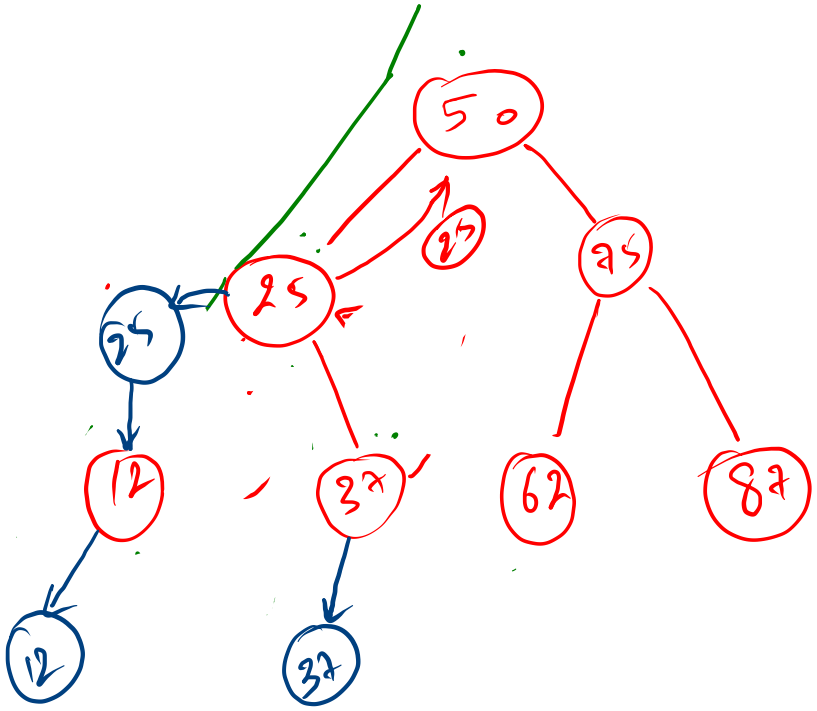

50
25
75
12
37
62
87
30
70



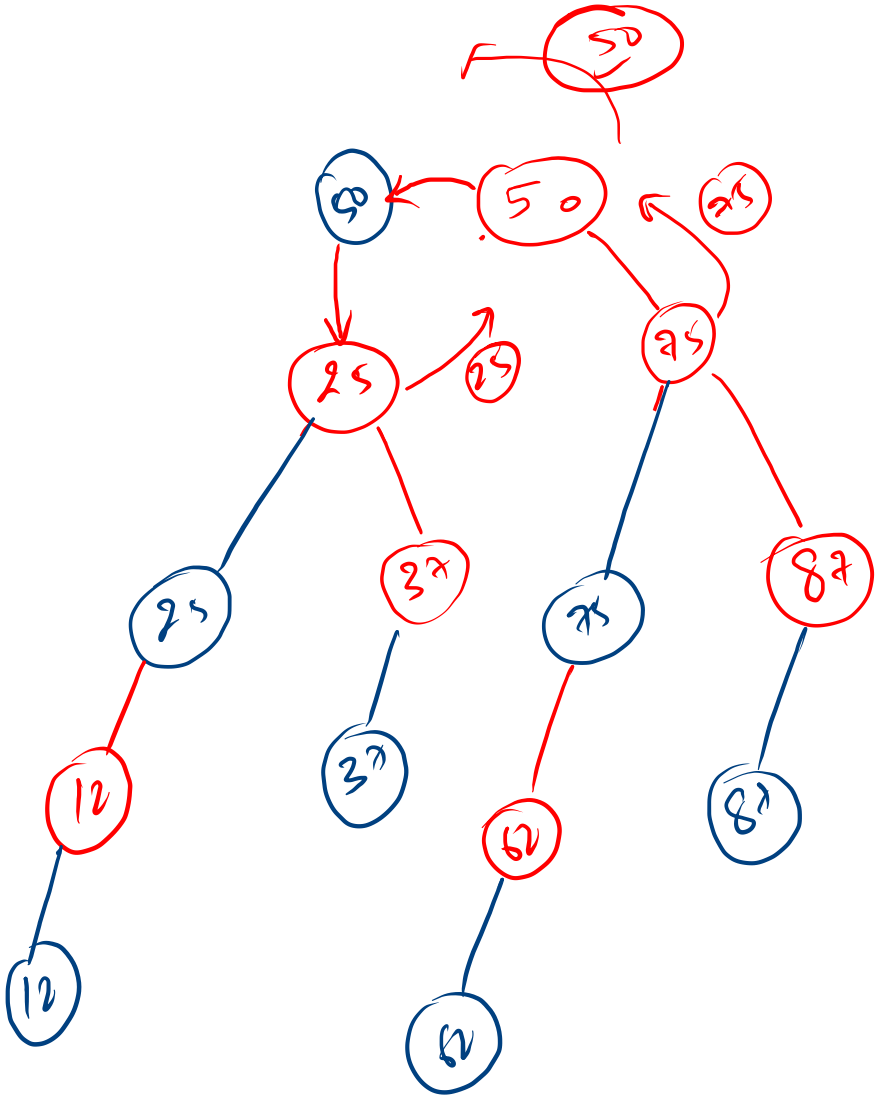

H.W.

```
public static void fun(Node node){
  if(node == null){
    return null;
  }
  Node lns = fun(node.lft.lft);
  Node rns = fun(node.right)
  Node clone = node.lft;
  clone.lft = null
  node.lft = lns;
  return node;
}
```





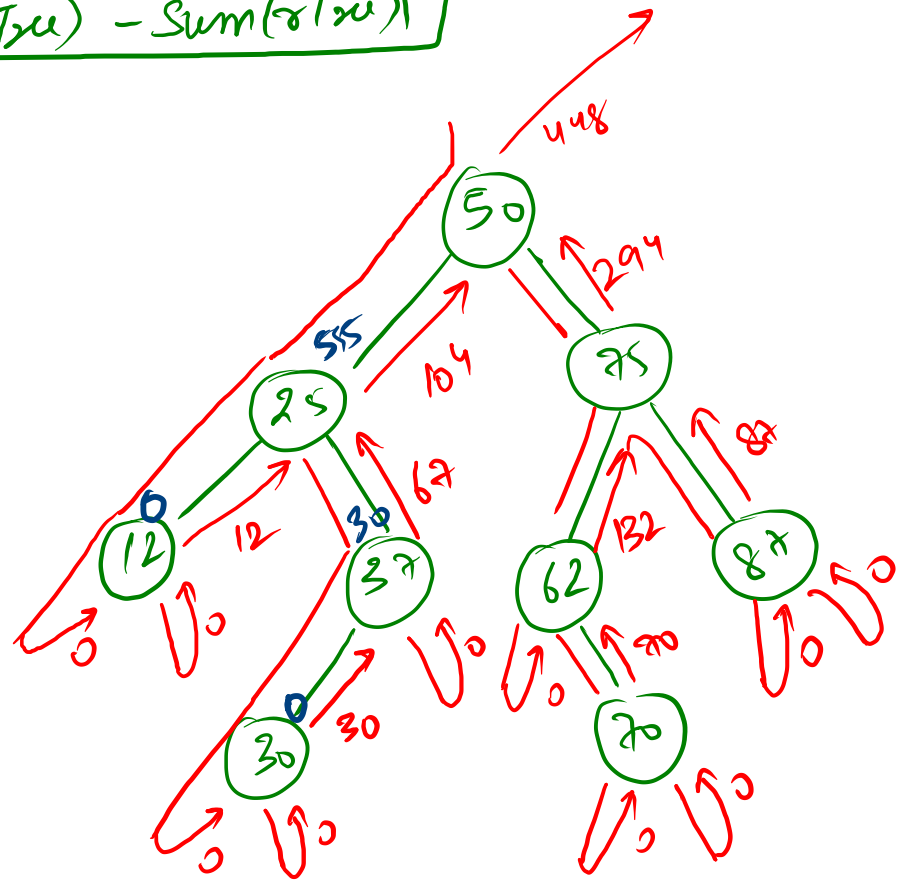

```java
public static Node createLeftCloneTree(Node node){
  if(node == null){
      return null;
  }
  Node lres = createLeftCloneTree(node.left);
  Node rres = createLeftCloneTree(node.right);

  Node clone = new Node(node.data,null,null);
  clone.left = lres;
  node.left = clone;

  return node;
}
```



```java
public static Node createLeftCloneTree(Node node){
  if(node == null){
      return null;
  }
  Node lres = createLeftCloneTree(node.left);
  Node rres = createLeftCloneTree(node.right);

  Node clone = new Node(node.data,null,null);
  clone.left = lres;
  node.left = clone;

  return node;
}
```

Tilt of ~~left~~ Tree = Sum ( Tilt of ~~left~~ every node )

Tilt of a node = |Sum(LTree) − Sum(RTree)|

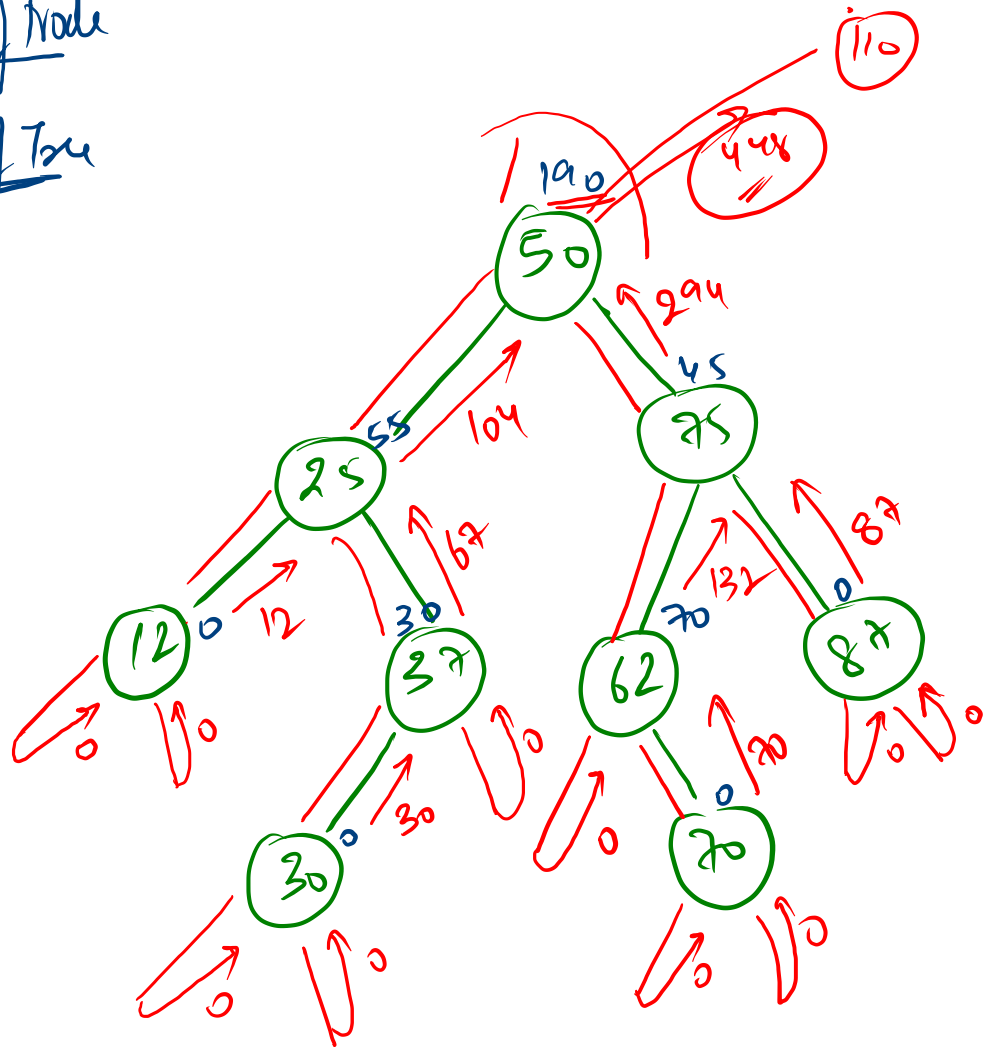



(Global variable hai)

Tilt = 0 + 0 + 0 + 30 + 55

Tilt of Node

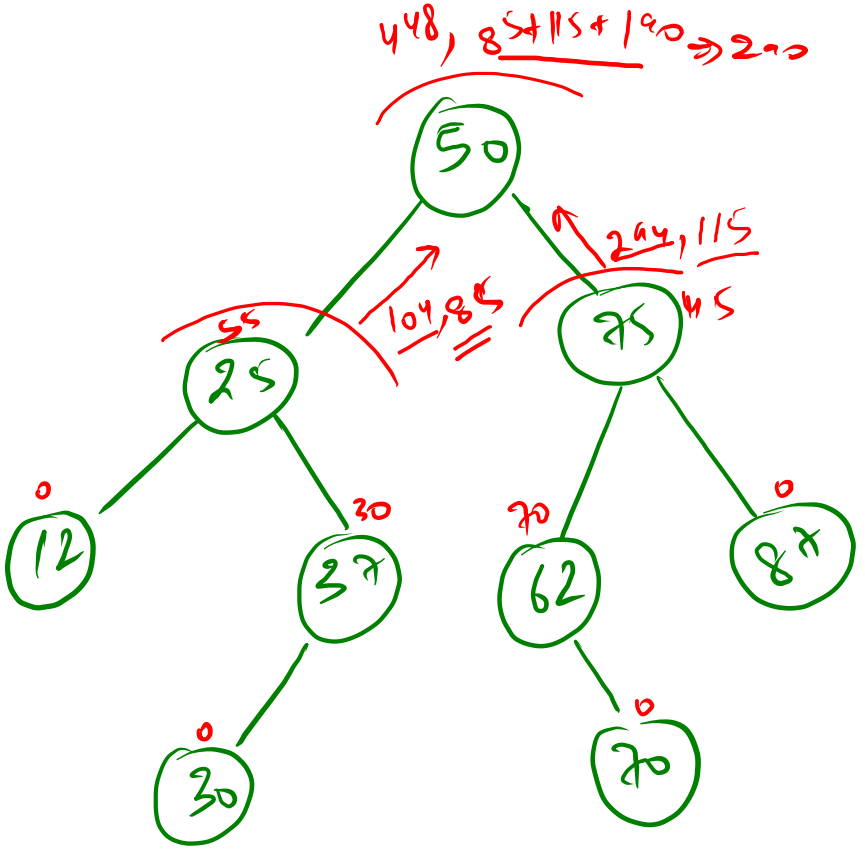
Tilt of Tree

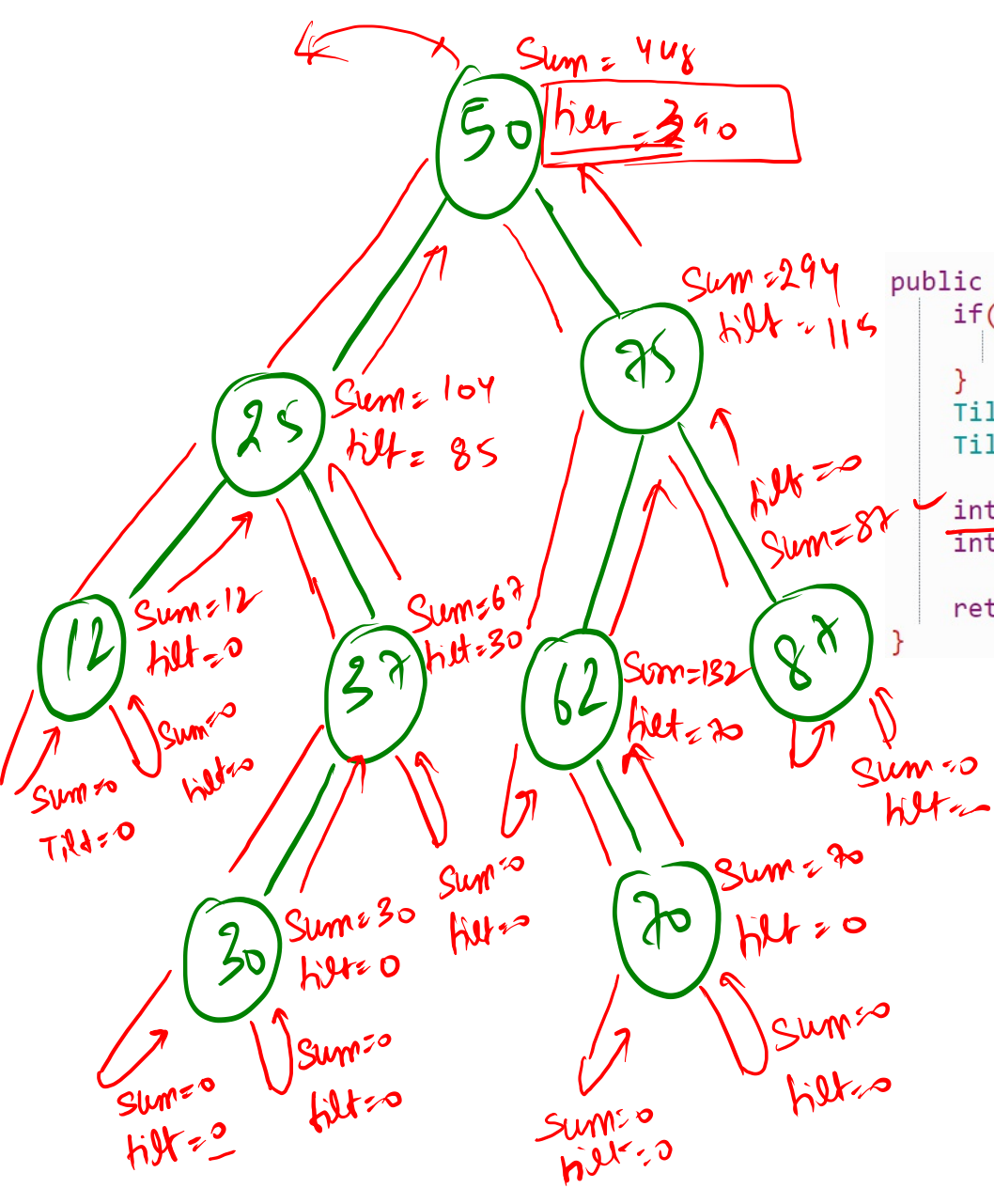



tiltOfTree = 0 + 0 + 0 + 30 + 55 + 0 + 70 + 0 + 45 + 190

→ 390

```
static int tiltOfTree = 0;
public static int tilt(Node node){
    if(node == null){
        return 0;
    }

    int lSum = tilt(node.left);
    int rSum = tilt(node.right);

    int tiltOfNode = Math.abs(lSum-rSum);
    tiltOfTree += tiltOfNode;

    return lSum+rSum+node.data;
}
```

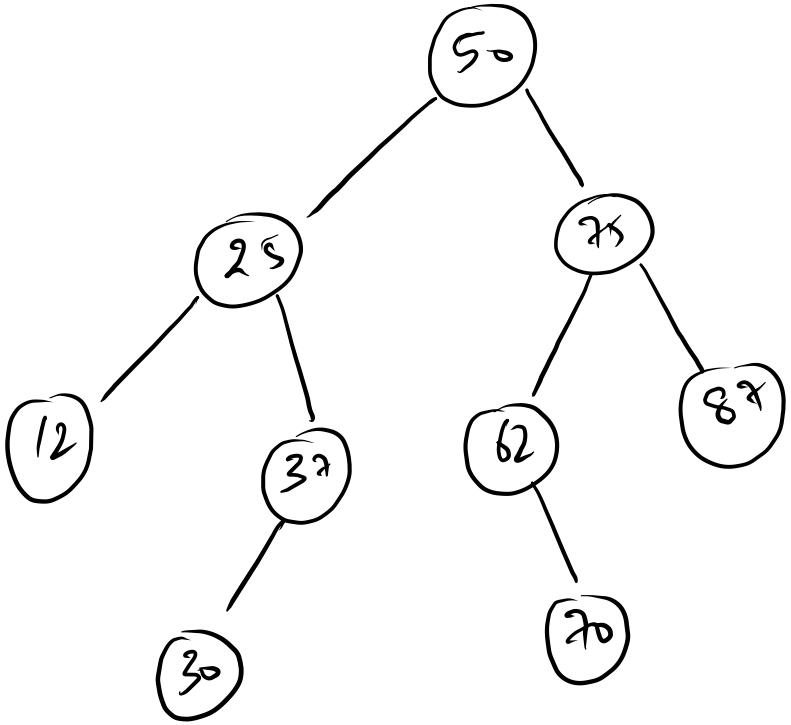

50
25
75
12
37
62
87
30
70
0
30
70
0
0
0



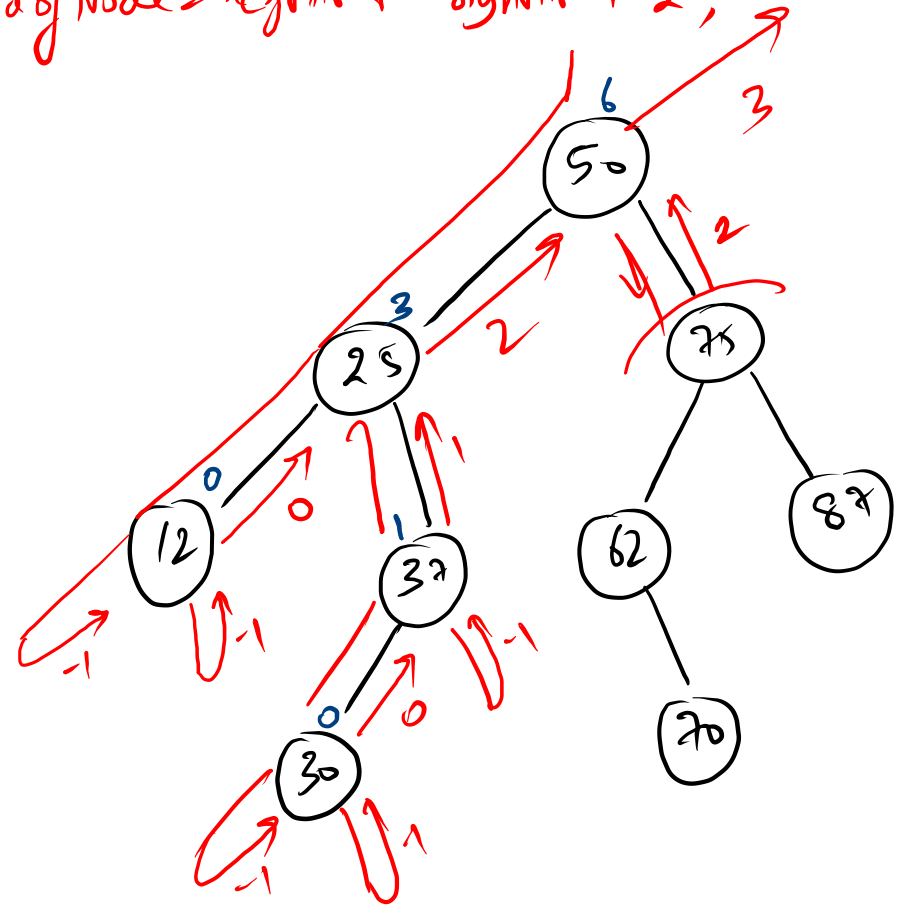

```
public static TiltSolver tilt1(Node node){
    if(node == null){
        return new TiltSolver(0,0);
    }
    TiltSolver lres = tilt1(node.left);
    TiltSolver rres = tilt1(node.right);

    int mySum = lres.sum+rres.sum+node.data;
    int tiltOfTree = lres.tiltOfTree+rres.tiltOfTree+Math.abs(lres.sum-rres.sum);

    return new TiltSolver(tiltOfTree,mySum);
}
```

Diameter → Max distance (on the basis of edges) b/w any two nodes in a binary Tree.

Dia = left ht + right ht + 2



dia of Node = leftHt + rightHt + 2;

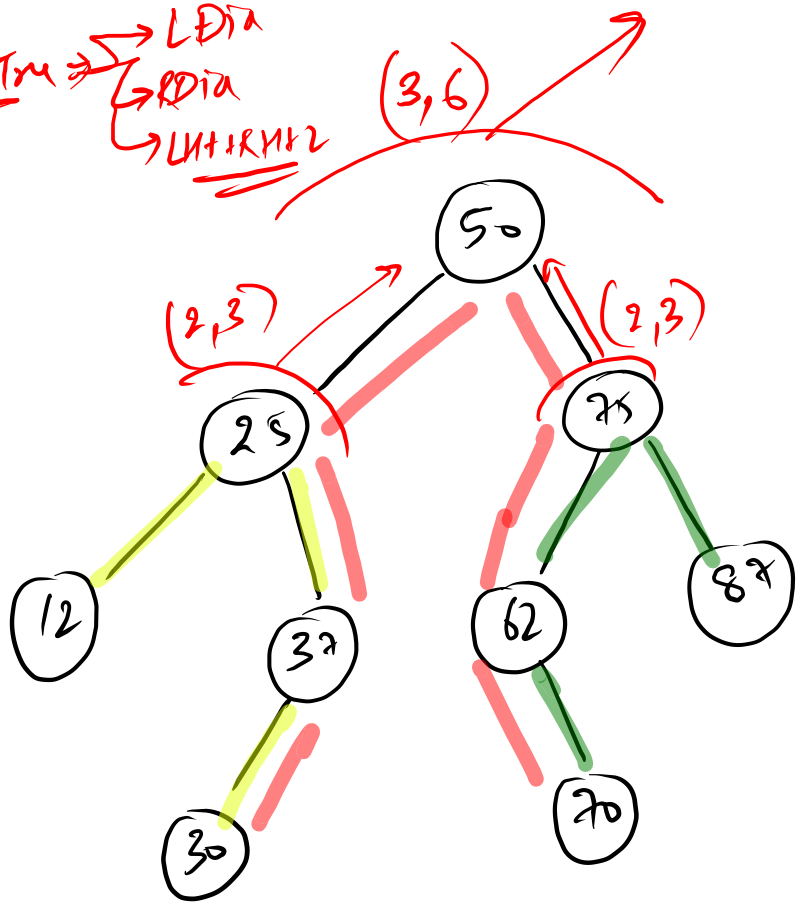



diaOfTree = ~~0~~ 6

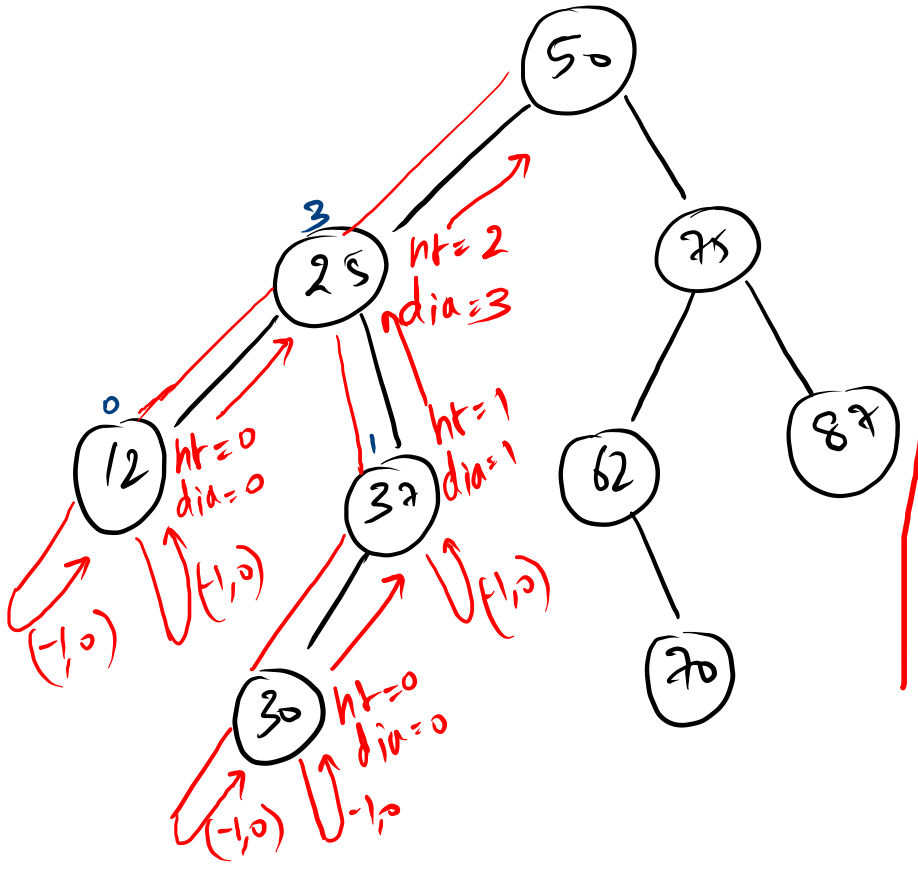
```
Node root = construct(arr);
diaOfTree = 0;
diameter1(root);
System.out.println(diaOfTree);
```

```
static int diaOfTree;
public static int diameter1(Node node) {
 if(node == null){
     return -1;
 }
 int leftHt = diameter1(node.left);
 int rightHt = diameter1(node.right);

 int diaOfNode = leftHt + rightHt + 2;
 if(diaOfNode > diaOfTree){
     diaOfTree = diaOfNode;
 }

 return Math.max(leftHt,rightHt)+1;
}
```

1. Hr
2. diaofTree ⇒ LDia, RDia, LHt+RHt+2




Diaofroot ⇒ 6




```java
public static DiaSolver diameter2(Node node){
    if(node == null){
        return new DiaSolver(-1,0);
    }
    DiaSolver lres = diameter2(node.left);
    DiaSolver rres = diameter2(node.right);

    int ht  = Math.max(lres.ht,rres.ht)+1;
    int diaOfNode = lres.ht+rres.ht+2;
    int diaOfTree = Math.max(diaOfNode,Math.max(lres.diaOfTree,rres.diaOfTree));

    return new DiaSolver(ht,diaOfTree);
}
```

# Binary Search Tree

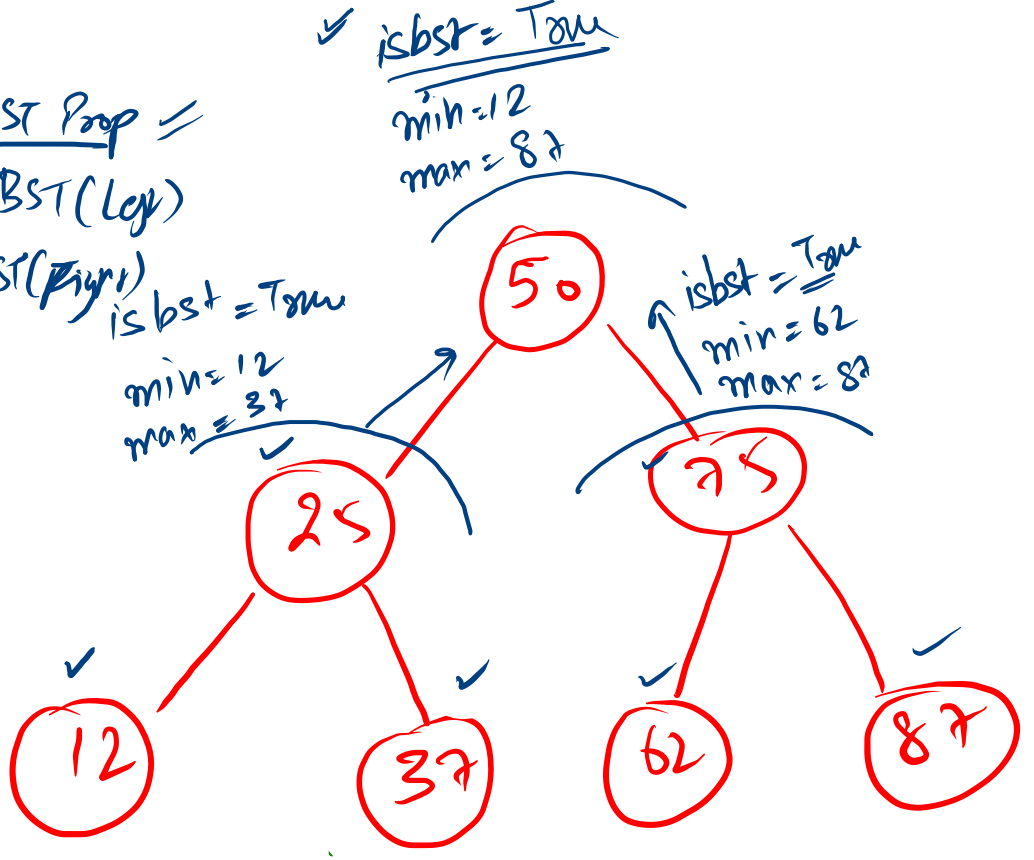
↳ all nods follow BST Poop

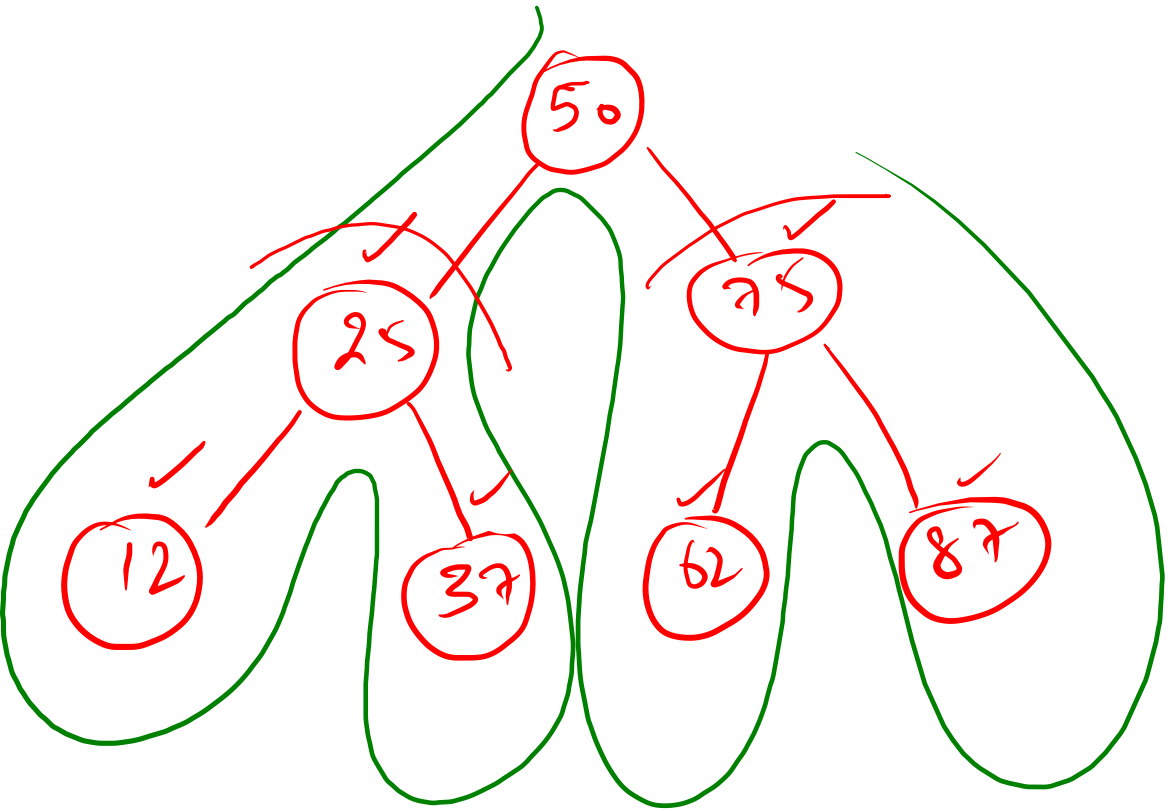
isBST() → BST Poop ✓
- isBST(left)
- isBST(right)

BST Poop:

left < n.data < right

Max(left) ≤ n.data < Min(right)

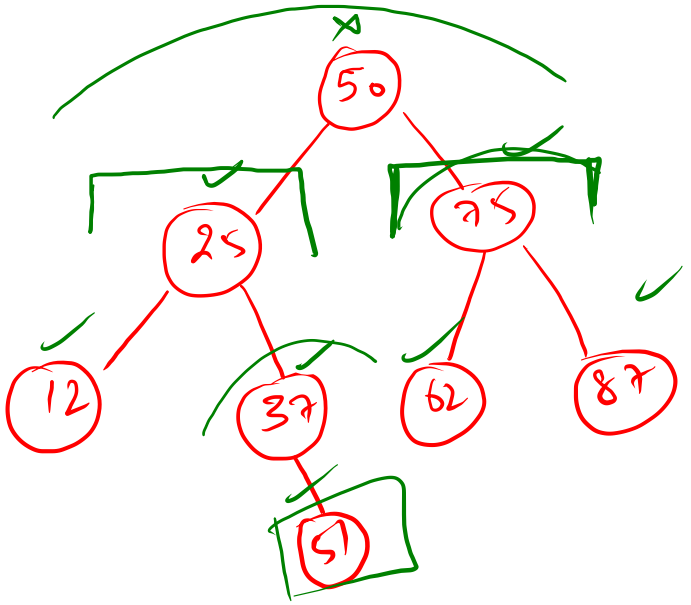





| | | Min | Max | isbst |
|---|---|---|---|---|
| | null | $+\infty$ | $-\infty$ | true |
| | 12 | 12 | 12 | true |
| | 37 | 37 | 37 | true |
| l | 25 | 12 | 37 | true |
| | 62 | 62 | 62 | true |
| | 87 | 87 | 87 | true |
| r | 75 | 62 | 87 | true |
| | 50 | 12 | 87 | true |

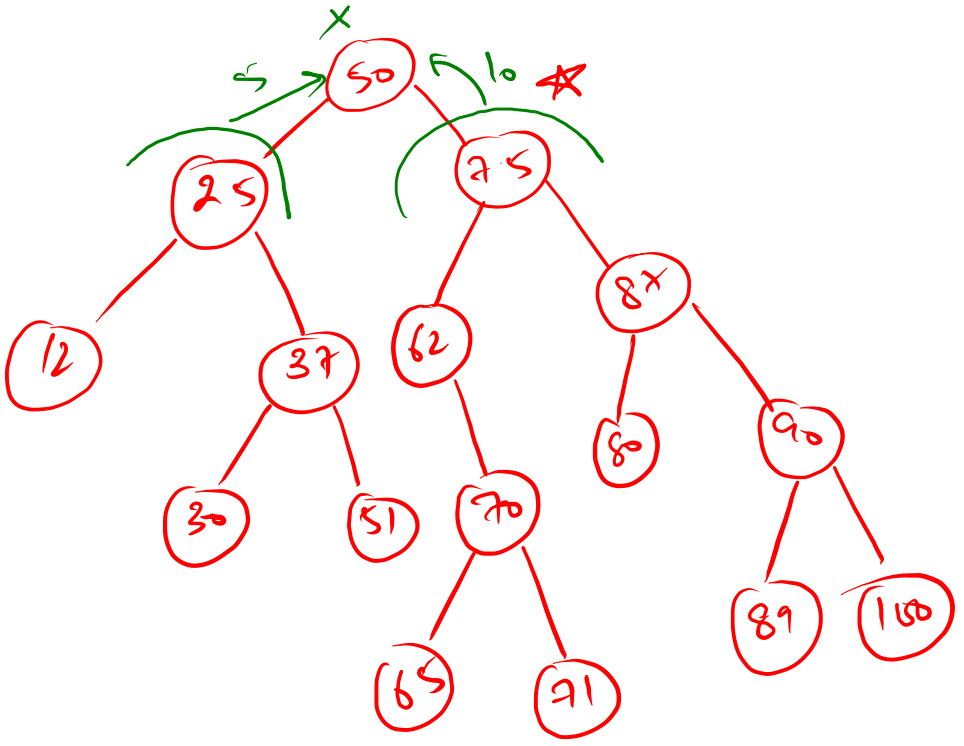
isbst (node)

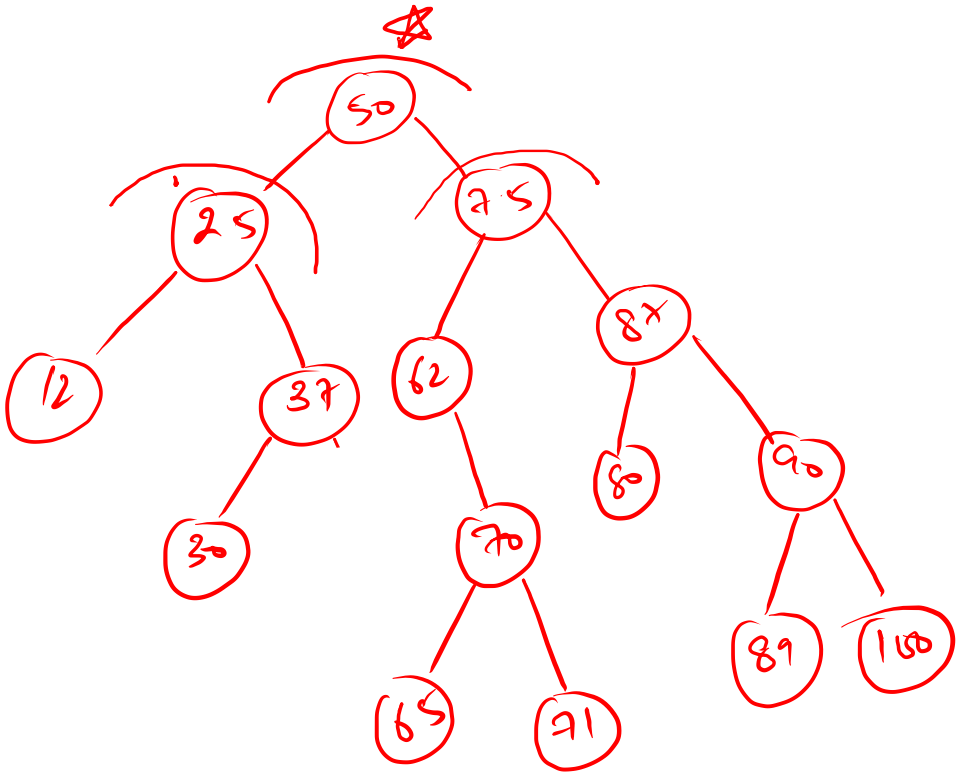
- isbst (left)
- isbst (right)
- Max(left) < n.data < Min(right)

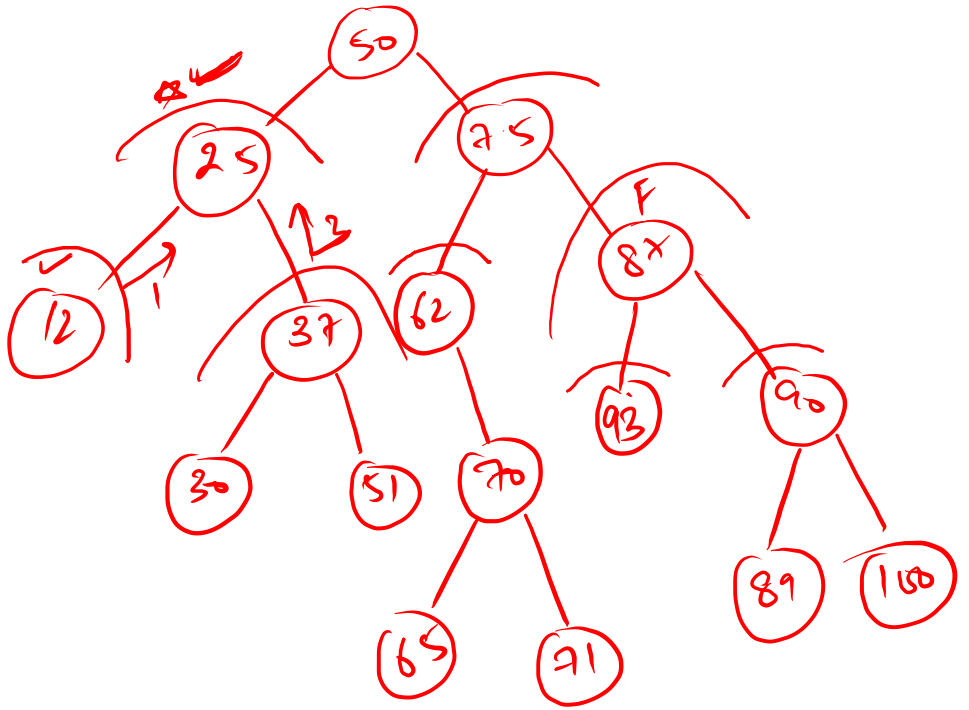



| | Node | Min | Max | isBst |
|---|---|---|---|---|
| | null | +∞ | -∞ | true |
| | 12 | 12 | 12 | true |
| | 51 | 51 | 51 | true |
| | 37 | 37 | 51 | true |
| l | 25 | 12 | 51 | true |
| | 62 | 62 | 62 | true |
| | 87 | 87 | 87 | true |
| r | 75 | 62 | 87 | true |
| | 50 | 12 | 87 | false |

```java
public static IsBSTSolver checkBst(Node node){
    if(node == null){
        return new IsBSTSolver(Integer.MAX_VALUE,Integer.MIN_VALUE,true);
    }
    IsBSTSolver lres = checkBst(node.left);
    IsBSTSolver rres = checkBst(node.right);

    int min = Math.min(node.data,Math.min(lres.min,rres.min));
    int max = Math.max(node.data,Math.max(lres.max,rres.max));
    boolean isbst = lres.isbst && rres.isbst && node.data > lres.max && node.data < rres.min;
    return new IsBSTSolver(min,max,isbst);
}
```

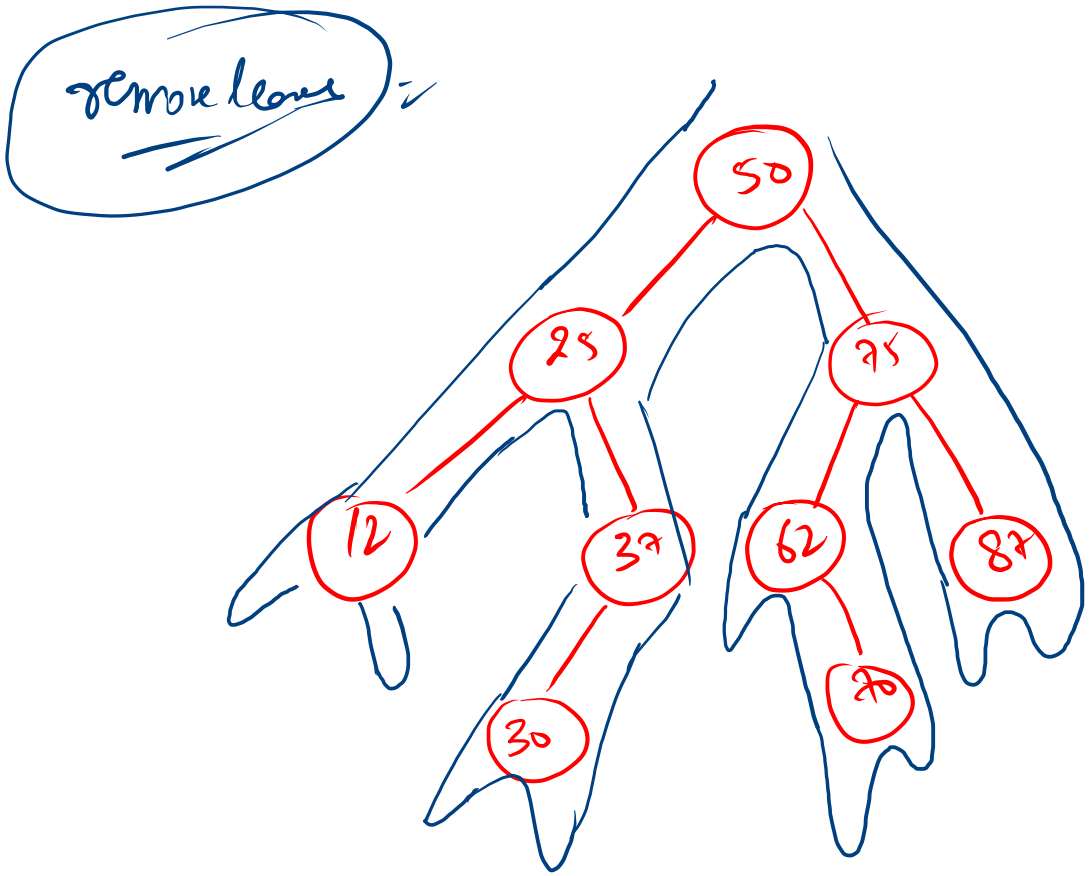

x
5
10
50
25
75
12
37
62
87
30
51
70
80
90
65
71
89
100

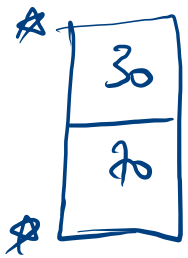

50
25
75
12
37
62
87
30
70
80
90
65
71
89
100



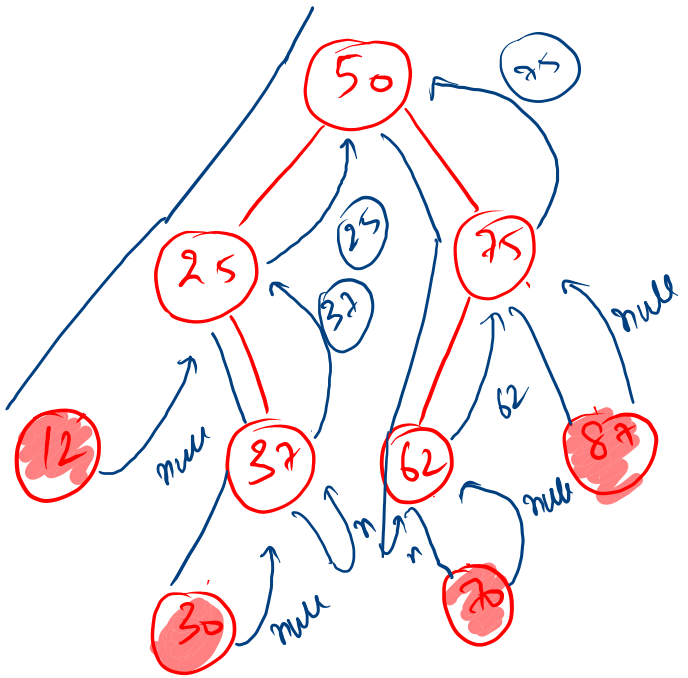

| | | Min | Max | isbst | lBSTNode | lBSTSize |
|---|---|---|---|---|---|---|
| | null | +∞ | -∞ | true | null | 0 |
| | 12 | 12 | 12 | true | 12 | 1 |
| | 30 | 30 | 30 | true | 30 | 1 |
| | 51 | 51 | 51 | true | 51 | 1 |
| | 37 | 30 | 51 | true | 37 | 3 |
| l | 25 | 12 | 51 | true | 12 51 | 5 |
| | 65 | 65 | 65 | true | 65 | 1 |
| | 71 | 71 | 71 | true | 71 | 1 |
| → | 70 | 65 | 71 | true | 70 | 3 |
| → | 62 | 62 | 71 | true | 62 | 4 |
| → | 80 | 80 | 80 | true | 80 | 1 |
| → | 89 | 89 | 89 | true | 89 | 1 |
| → | 100 | 100 | 100 | true | 100 | 1 |
| → | 90 | 89 | 100 | true | 90 | 3 |
| | 87 | 80 | 100 | true | 90 | 3 |
| r | 75 | 62 | 100 | true | 62 | 4 |
| | 50 | 12 | 100 | false | 25 | 5 |

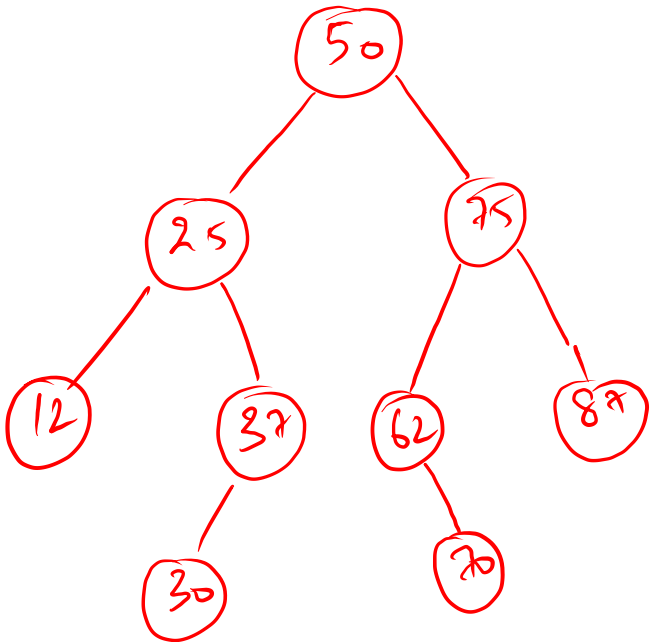
```
if (isbst){

lbstNode = node;

lbst size = lres.size + r.size + 1;

}
```



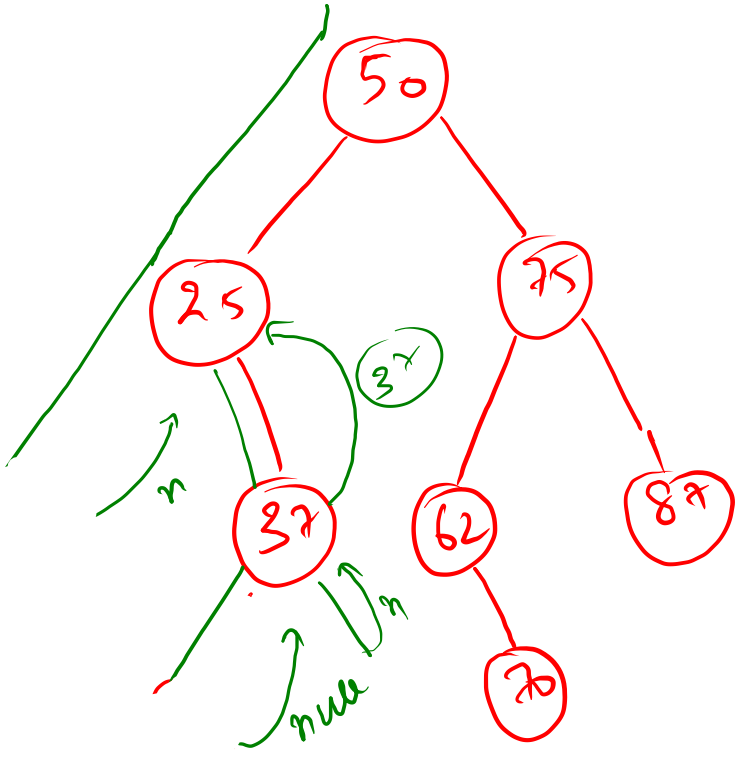




```java
public static void printSingleChildNodes(Node node){
    if(node == null){
        return;
    }

    if(node.right == null && node.left != null){
        System.out.println(node.left.data);
    }else if(node.right != null && node.left == null){
        System.out.println(node.right.data);
    }

    printSingleChildNodes(node.left);
    printSingleChildNodes(node.right);
}
```

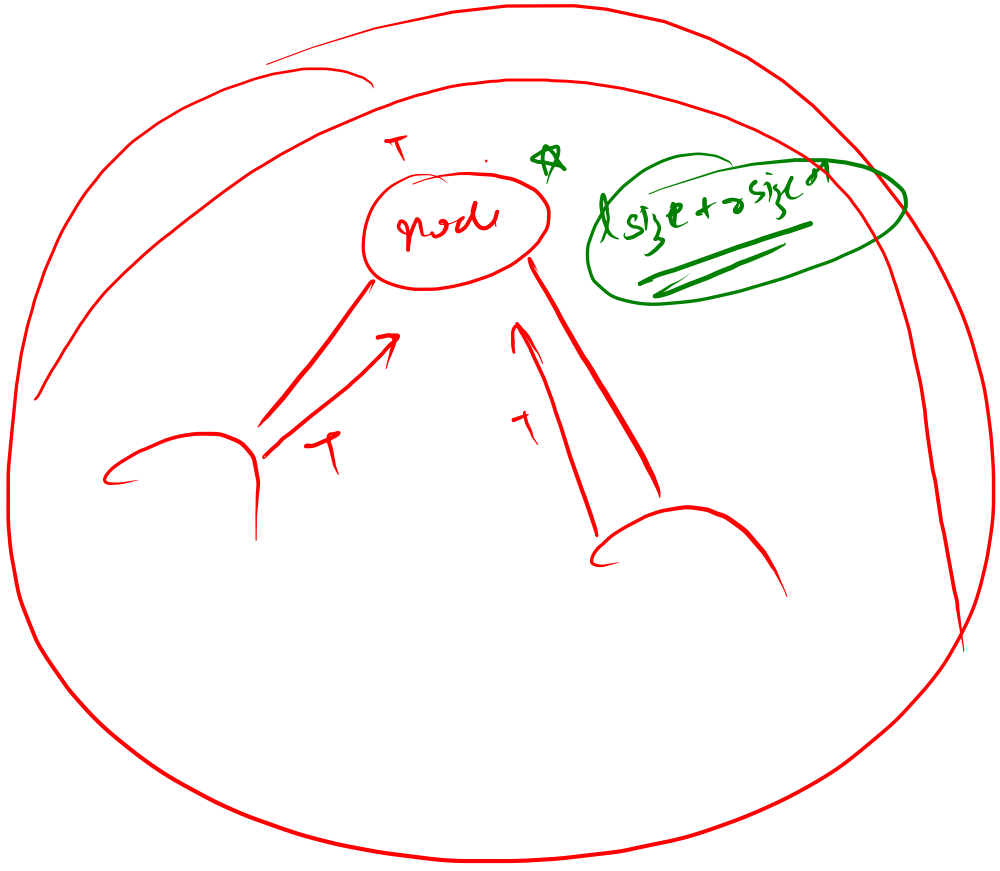

50
75
25
25
37
75
null
12
null
37
62
62
87
null
n
n
30
null
70

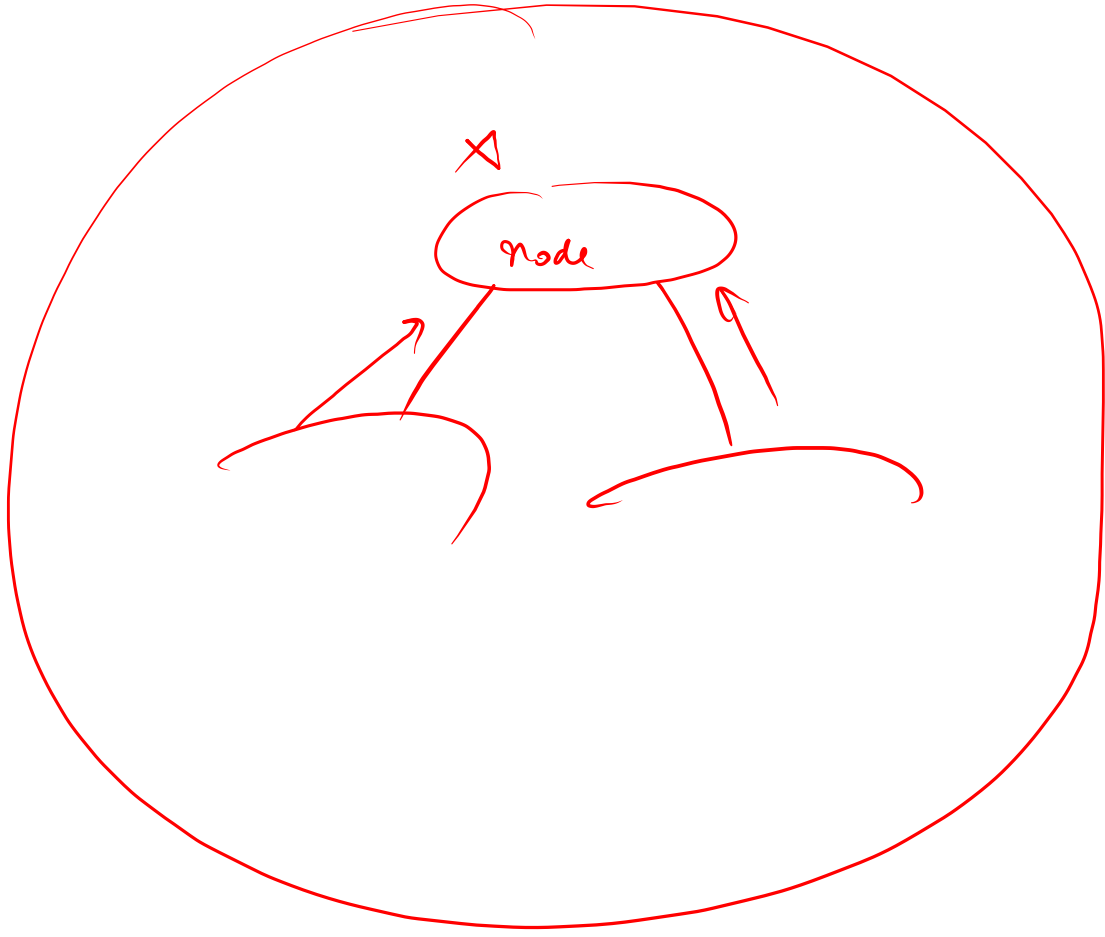

50
25
75
12
37
62
87
30
70

→ largest bst subtree




```java
public static Node removeLeaves(Node node){
  if(node == null){
      return null;
  }

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

  node.left = removeLeaves(node.left);
  node.right = removeLeaves(node.right);

  return node;
}
```

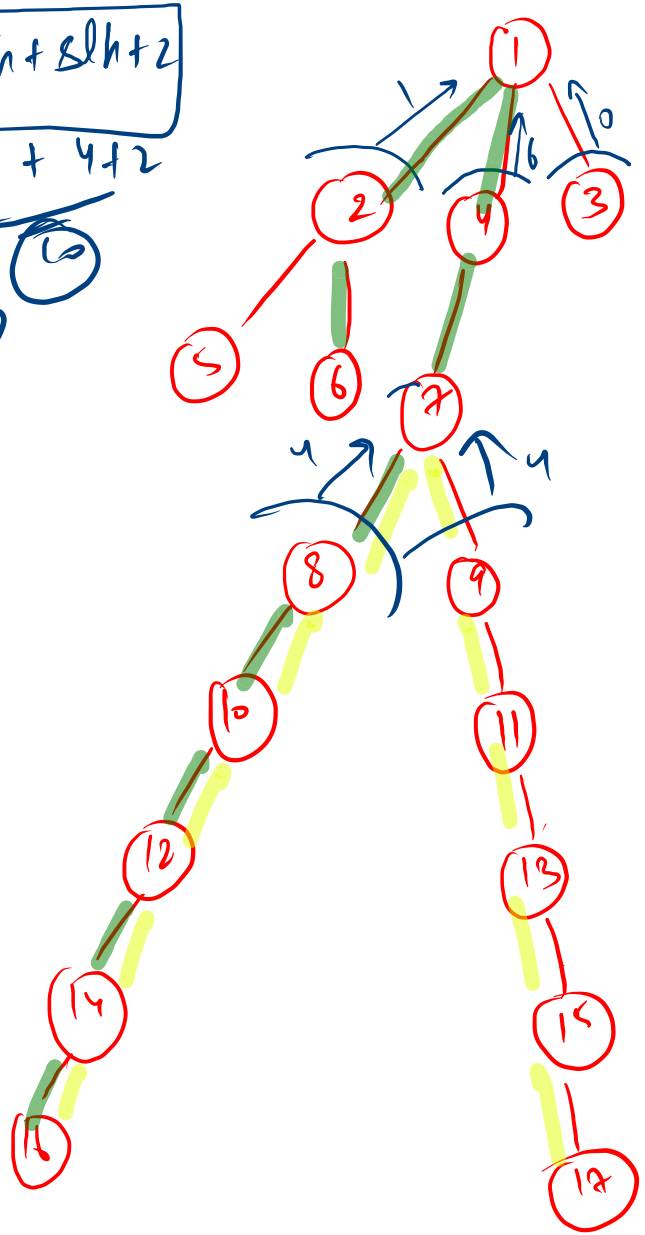

T
node
size+size
T
T


node

Dia ⇒ lh + slh + 2

4 + 4 + 2

10




**Dia**

↳ Maximum distance b/w any two nodes.

6 - 16 ⇒ ?

16 - 17 ⇒ 10

Dia = lh + slh + 2

Dia mein ⇒ 10