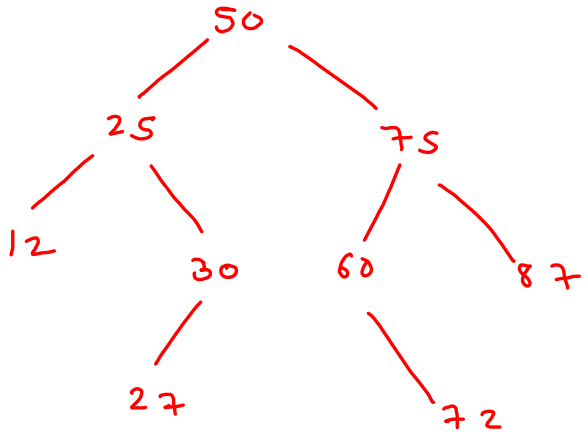
BST : binary search tree

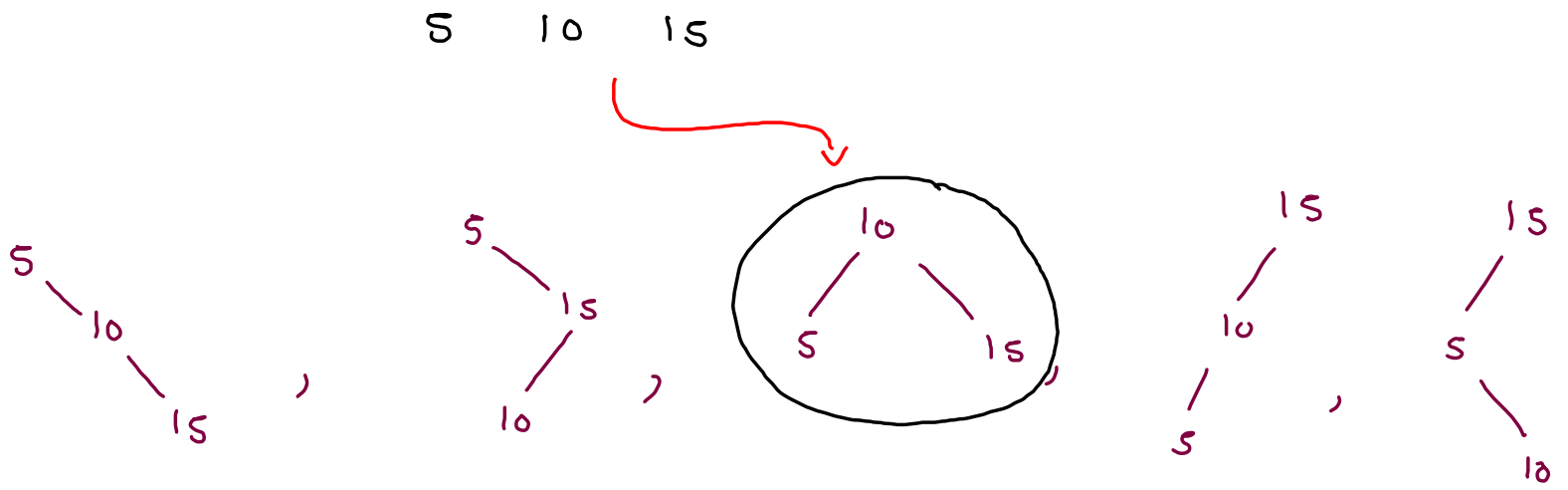


∀ nodes :

all nodes in left subtree < node.data < all nodes in right subtree

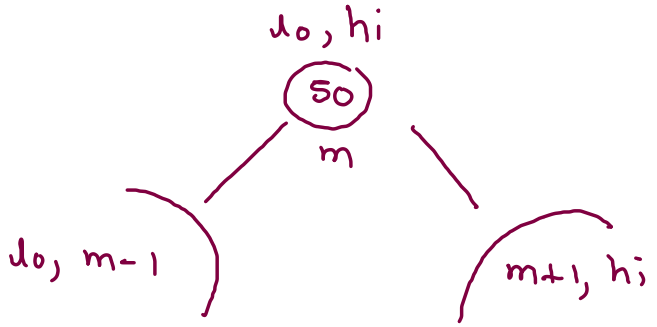
Convert Sorted Array to Binary Search Tree

height balanced BST



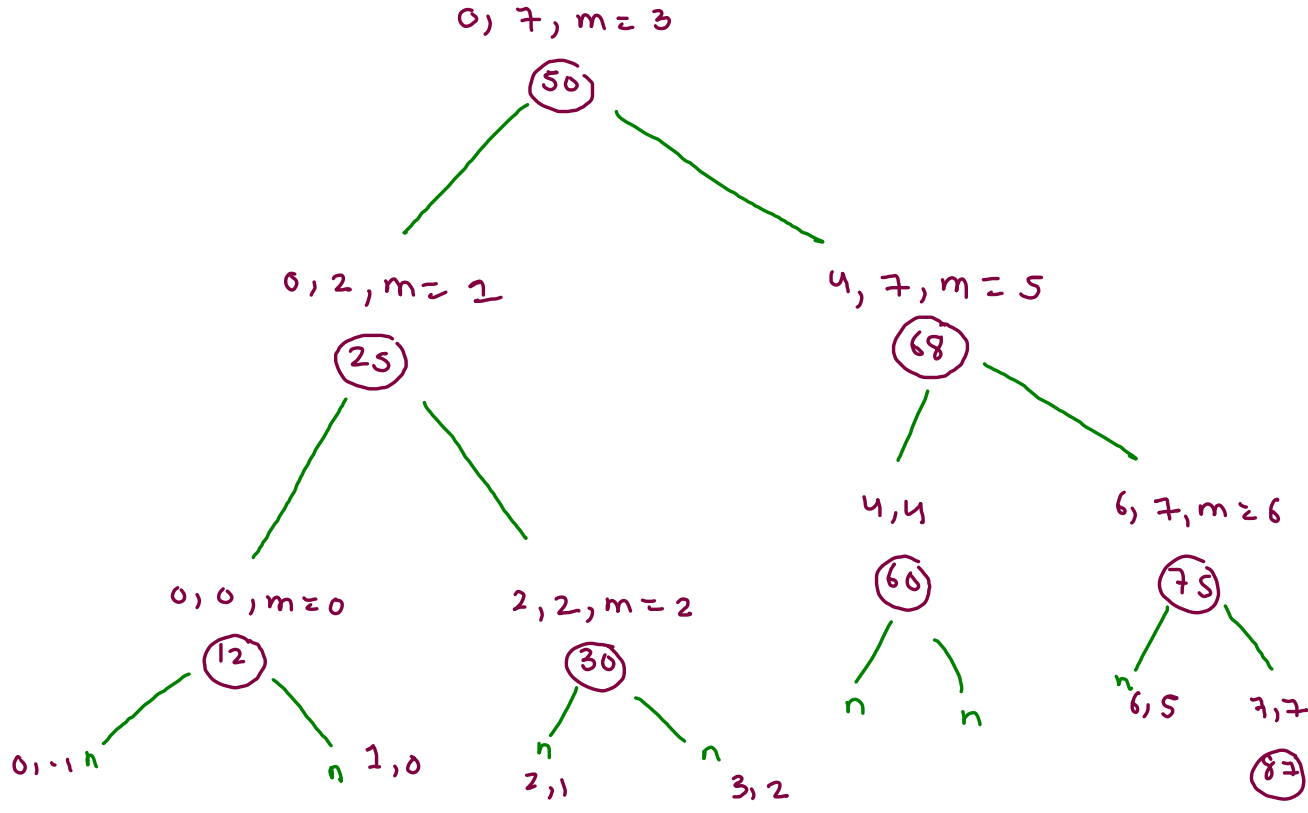
12	25	30	50	60	68	75	87
0	1	2	3	4	5	6	7

lo m hi

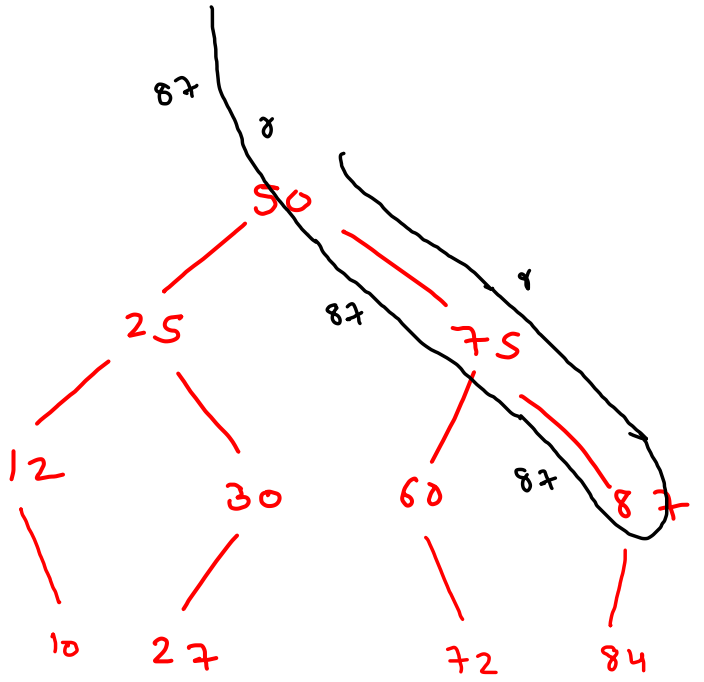


12	25	30	50	60	68	75	87
0	1	2	3	4	5	6	7

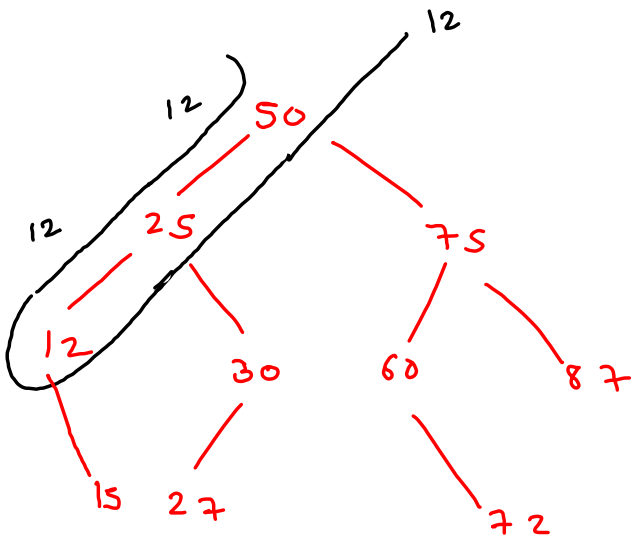
```java
public TreeNode helper(int lo,int hi,int[]nums) {

    int mid = (lo + hi)/2;

    TreeNode node = new TreeNode(nums[mid]);

    node.left = helper(lo,mid-1,nums);
    node.right = helper(mid+1,hi,nums);

    return node;
}
```



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

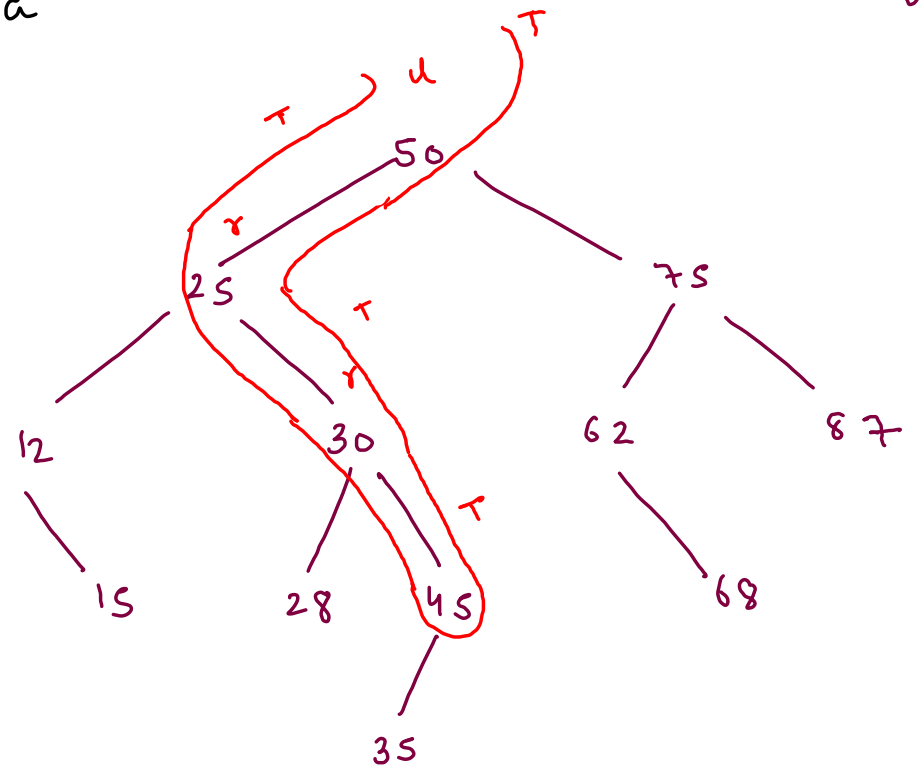


```java
public static int min(Node node) {
    if(node.left == null) {
        return node.data;
    }
    else {
        int lans = min(node.left);
        return lans;
    }
}
```

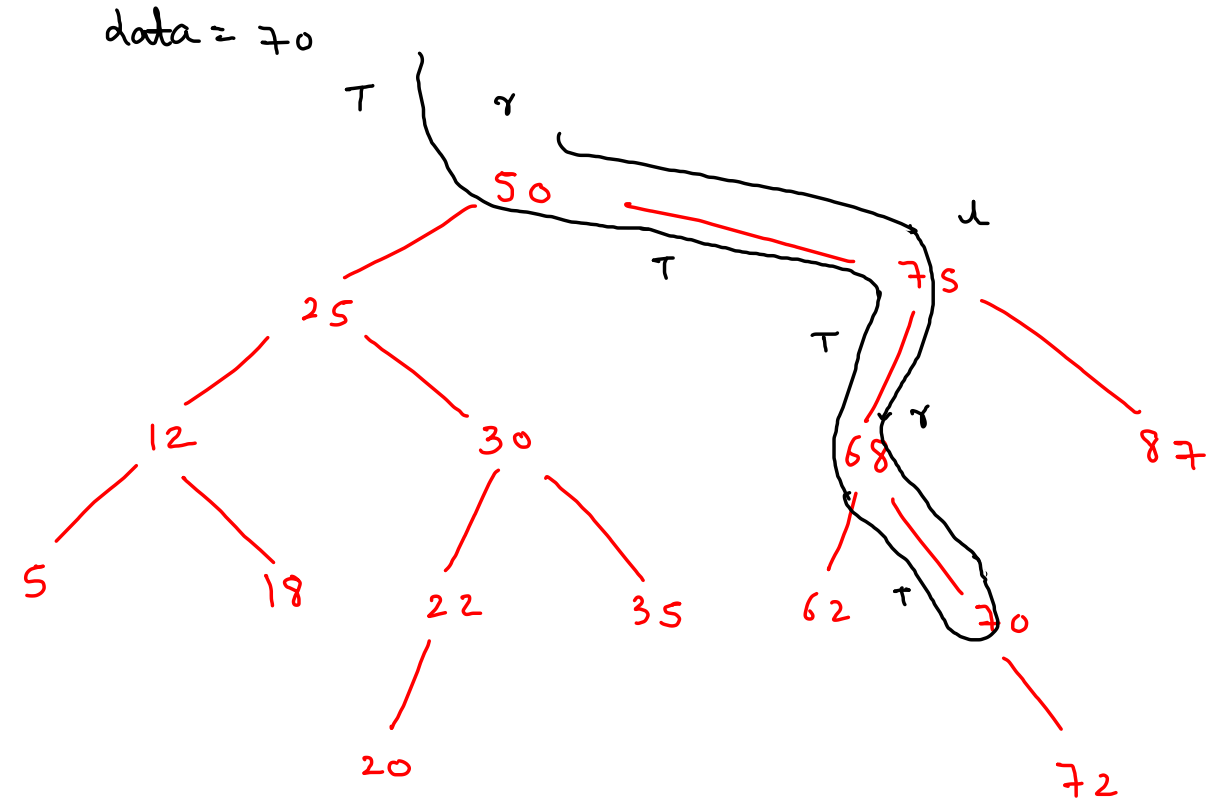


Find

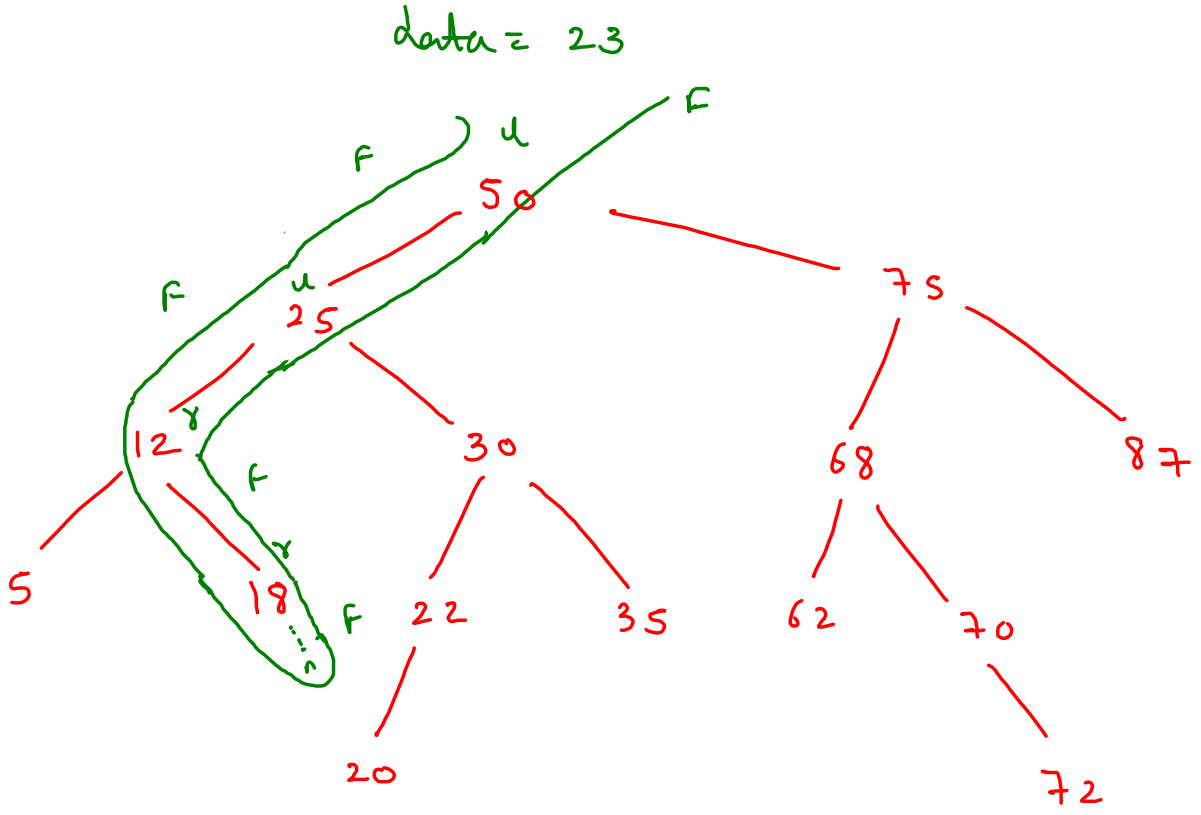
target = 45

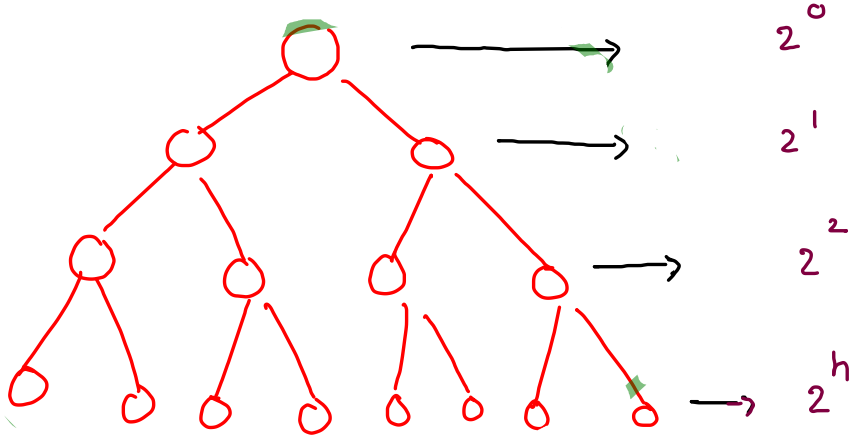


```java
public static boolean find(Node node, int data){
    if(node == null) {
        return false;
    }

    if(node.data == data) {
        return true;
    }
    else if(node.data < data) {
        boolean rans = find(node.right,data);
        return rans;
    }
    else {
        boolean lans = find(node.left,data);
        return lans;
    }
}
```



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

    if(node.data == data) {
        return true;
    }
    else if(node.data < data) {
        boolean rans = find(node.right,data);
        return rans;
    }
    else {
        boolean lans = find(node.left,data);
        return lans;
    }
}
```





(i) full binary tree

2, 0 child count is possible.

$$n = 2^0 + 2^1 + \dots\dots + 2^h$$

GP $a = 1$, $r = 2$, $t = h+1$

$$S = \frac{a(r^t - 1)}{r - 1} = \frac{1(2^{h+1} - 1)}{1}$$

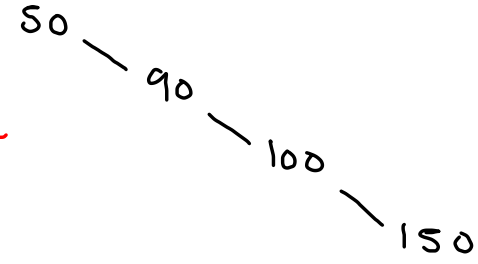
$$n = 2^{h+1} - 1$$

$$n = 2^{h+1}$$

$$h \approx \log_2 n$$

$$\log_2 n = h + 1$$

$h \approx \log_2 n$

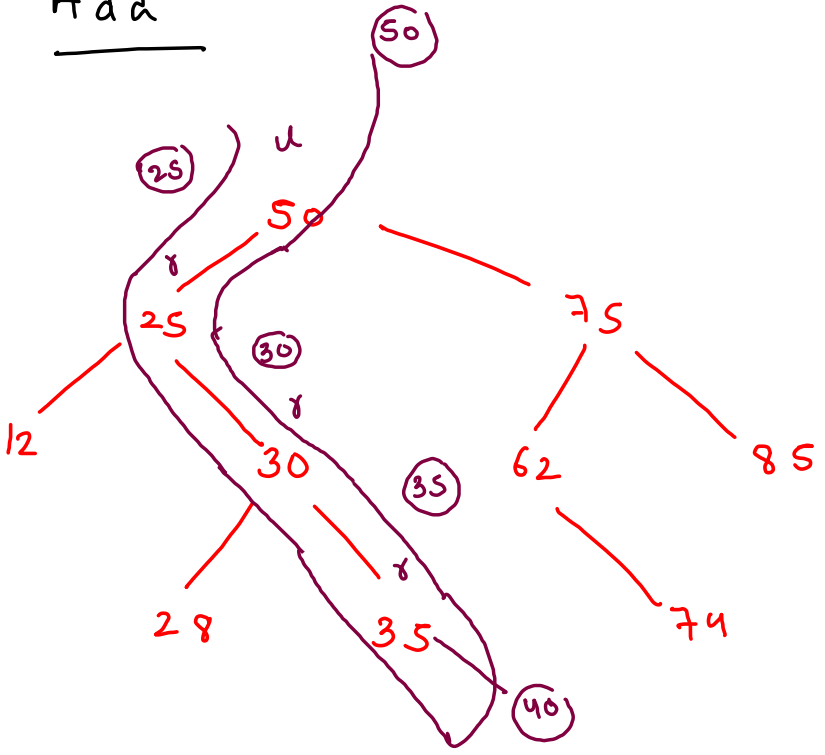


$h \approx n$ ←

BST find : $\log_2 n$ ∝

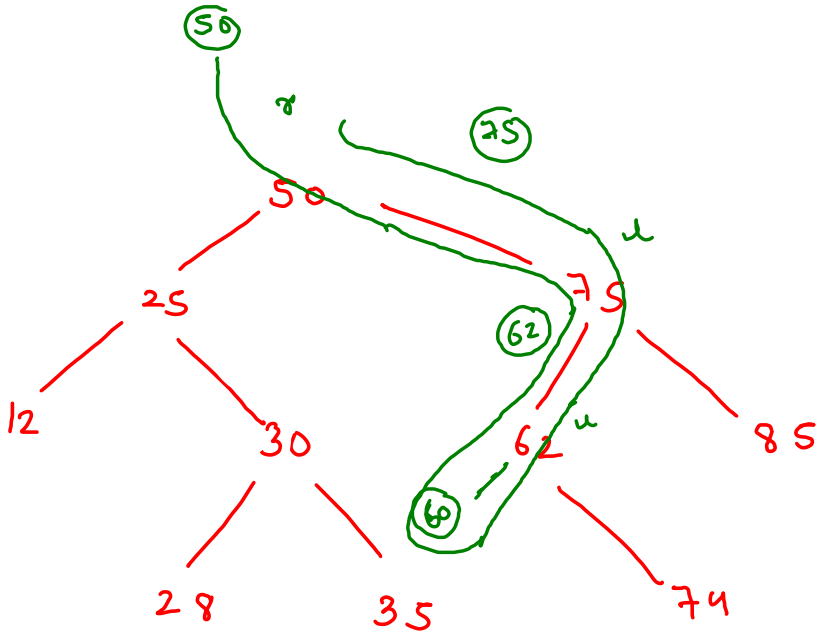
BST find : height

Add



data = 40

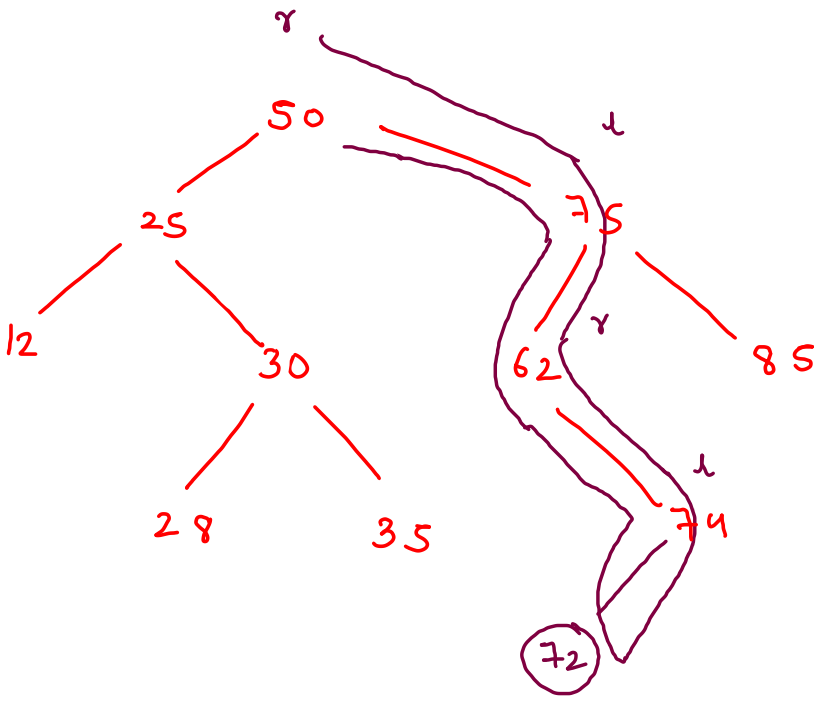
```
public static Node add(Node node, int data) {
    if(node == null) {
        return new Node(data);
    }

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

    return node;
}
```



data : 60

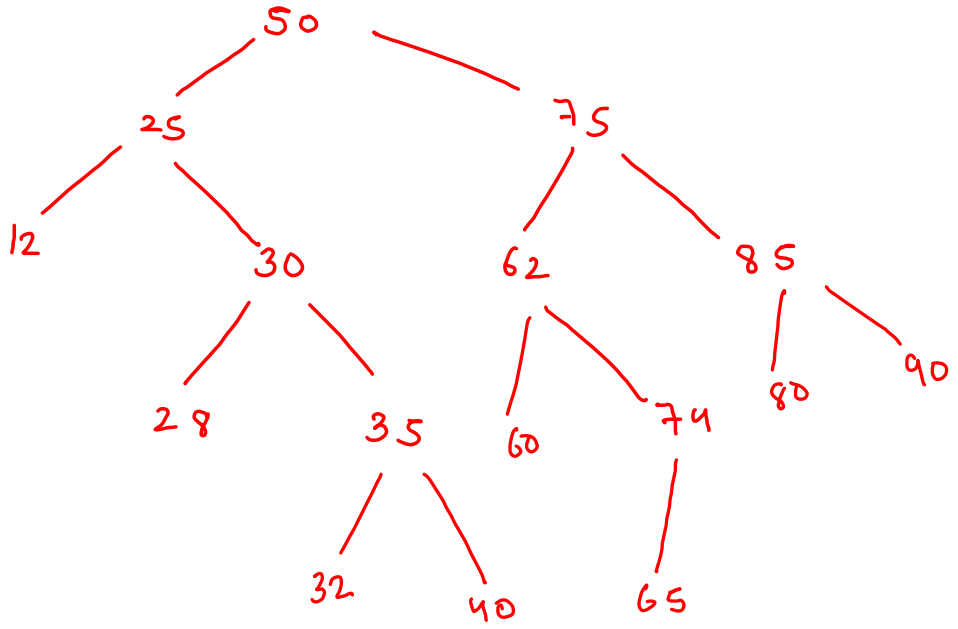
```
public static Node add(Node node, int data) {
        if(node == null) {
            return new Node(data);
        }

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

        return node;
}
```

data = 72




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

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

    return node;
}
```

Remove Node From Bst



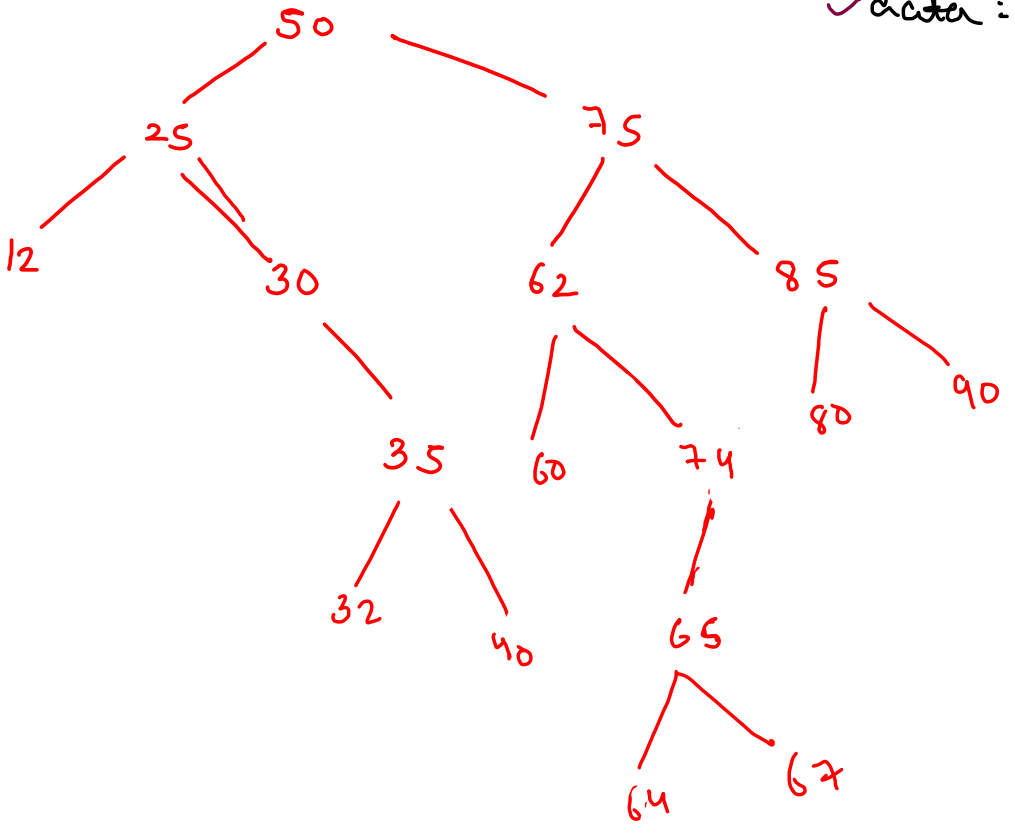
node to be removed:

(i) no child: return null

(ii) single child: return single child instead of me

(iii) two child: a) replace node.data with dmax or dmin.

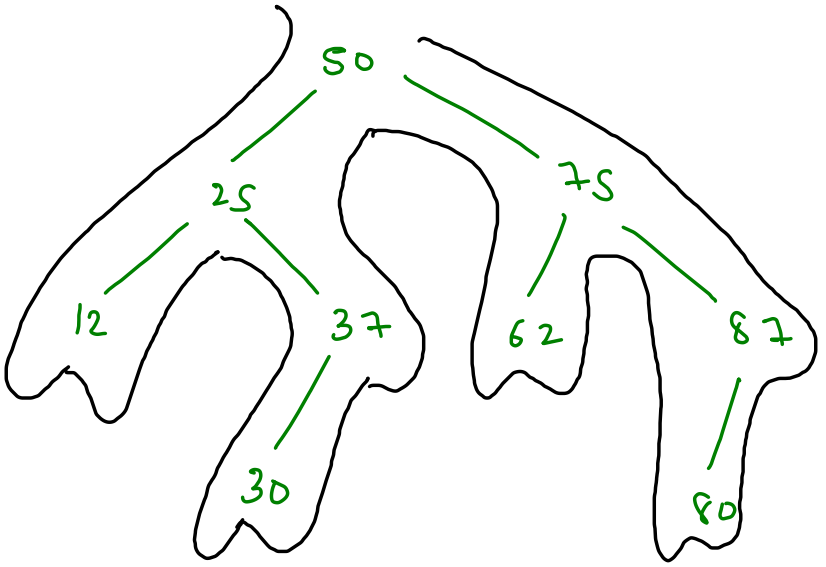
b) delete dmax.



- data = 32
- data = 30
- data = 75
- data = 50

```java
public static Node remove(Node node, int data) {
    if(node == null) {
        //data is not present in tree
        return null;
    }

    if(node.data < data) {
        node.right = remove(node.right,data);
    }
    else if(node.data > data) {
        node.left = remove(node.left,data);
    }
    else {
        //no child
        if(node.left == null && node.right == null) {
            return null;
        }
        //single child
        else if(node.left == null) {
            //node has only one child - right child
            return node.right;
        }
        else if(node.right == null) {
            //node has only one child - left child
            return node.left;
        }
        //both childs
        else {
            int lmax = max(node.left);
            node.data = lmax;
            node.left = remove(node.left,lmax);
        }
    }


    return node;
}
```

Inorder:



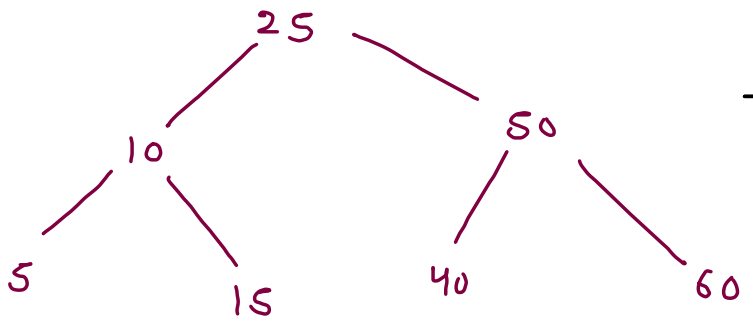
(*) Inorder of BST is always sorted.

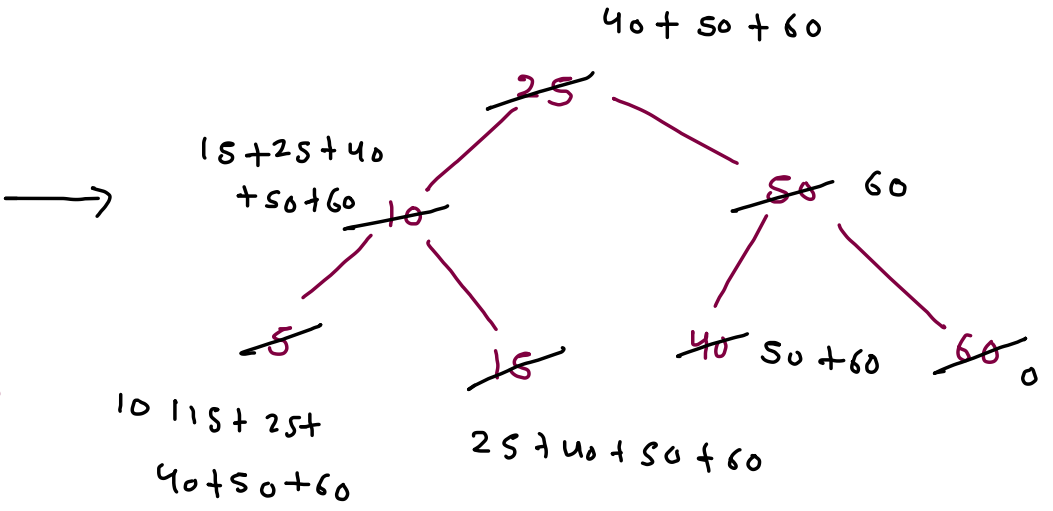
12 25 30 37 50 62 75 80 87

Replace With Sum Of Larger

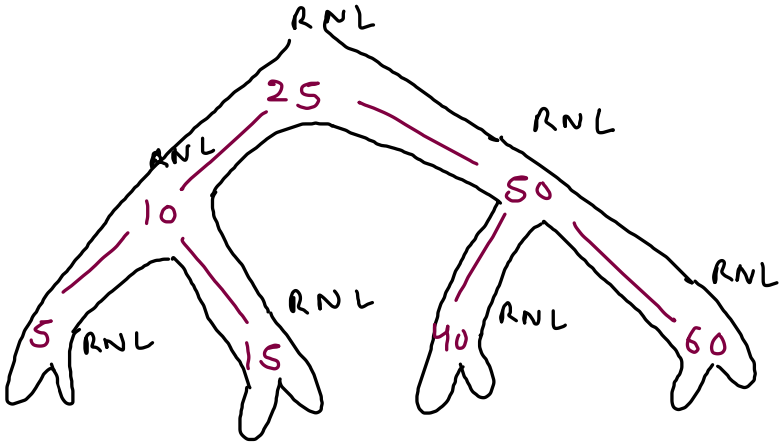
LNR -> regular inorder

RNL -> reverse inorder





60 50 40 25 15 10 5



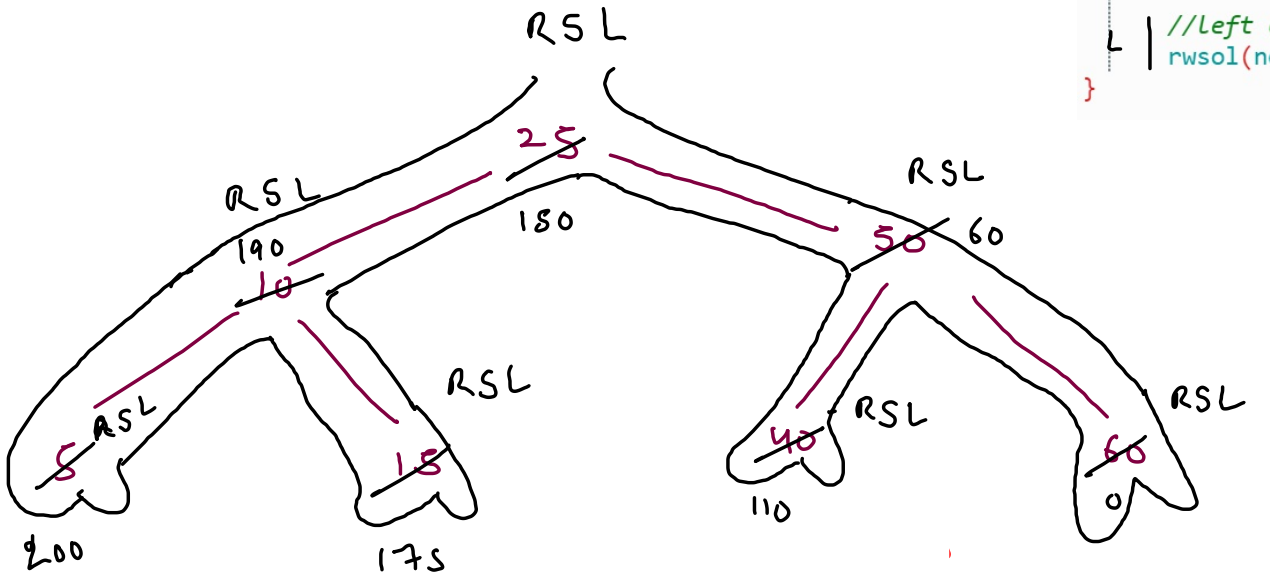
RNL

60 50 40 25 15 10 5

	regular	reverse
pre	NLR	NRL
In	LNR	RNL
post	LRN	RLN

Sum = 0 + 60 + 50 + 40 + 25 + 15 + 5

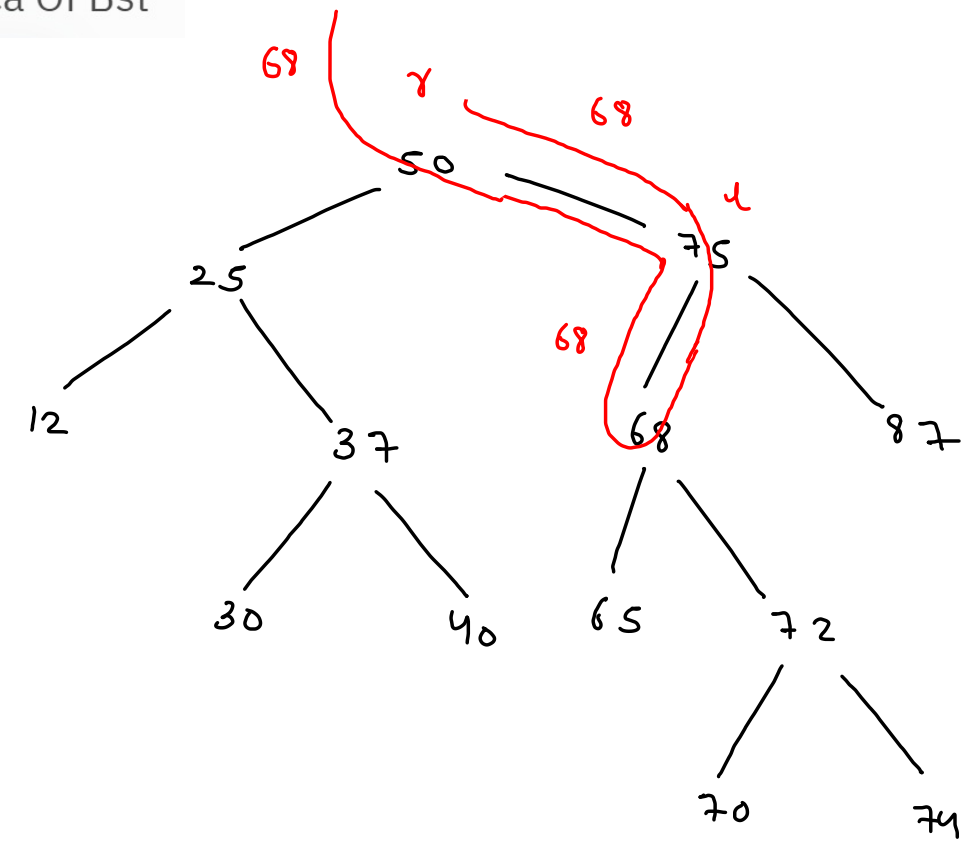
```java
static int sum = 0;
public static void rwsol(Node node){
    if(node == null) {
        return;
    }

    //right call
R   rwsol(node.right);

    //self work
S   int temp = node.data;
    node.data = sum;
    sum += temp;

    //left call
L   rwsol(node.left);
}
```



RNL

(reverse - inorder)

Lca Of Bst



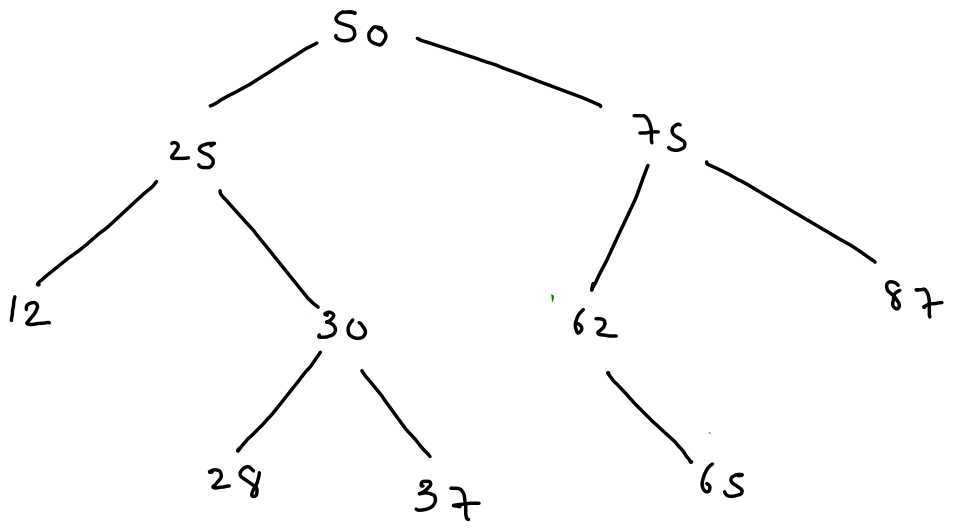
d1 = 65

d2 = 70

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

Print In Range

LSR



$d_0 = 25$

$h_i = 64$