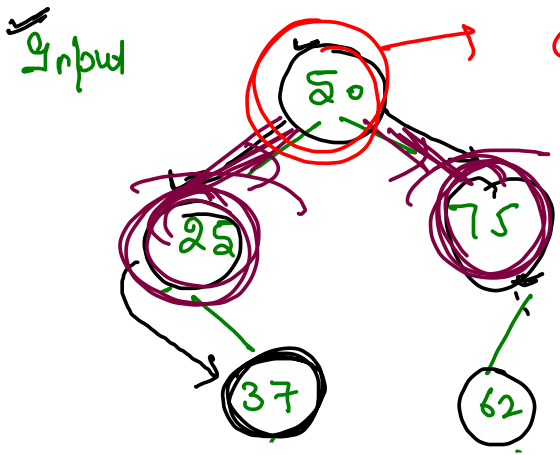
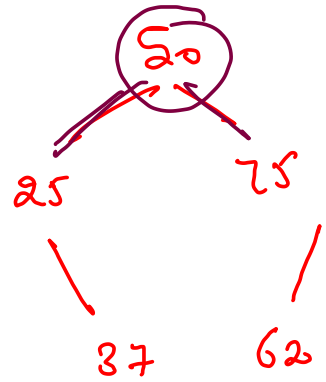
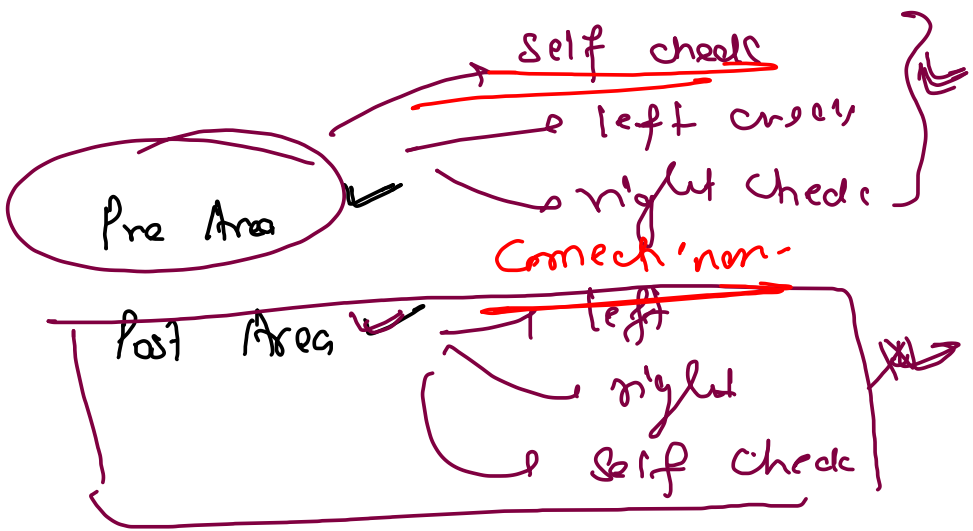
Remove leaves



Checking - pre-

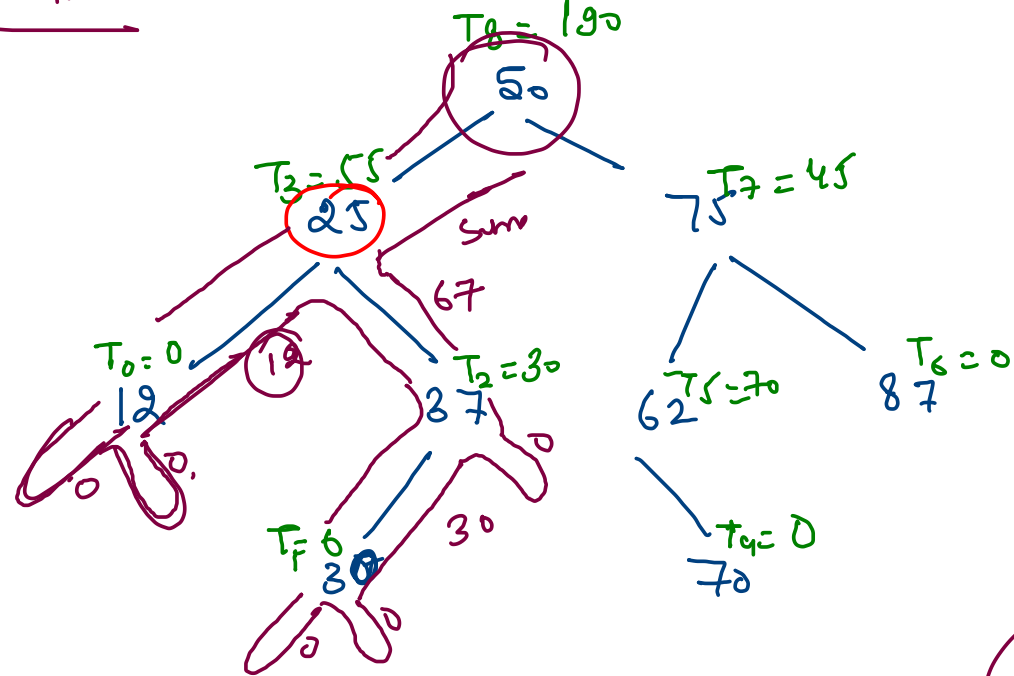
Connection - post





E — Avneet
Rupnarayan
Tanu
Diksha
Prashant

Tilt of tree:



Tilt of tree = $T_0 + T_1 + T_2 + \dots + T_8$

$= 390$

$T_0 + T_1 + T_2 + 3$

add in Static variable

return sum

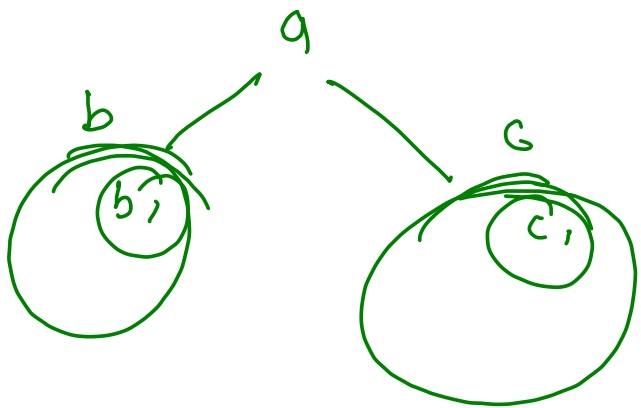
Binary Search Tree } → O(n)

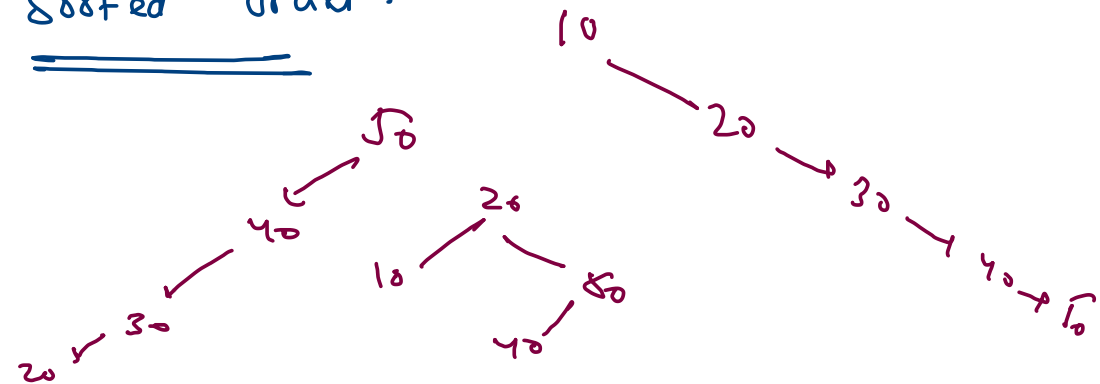
① Properties of BST-

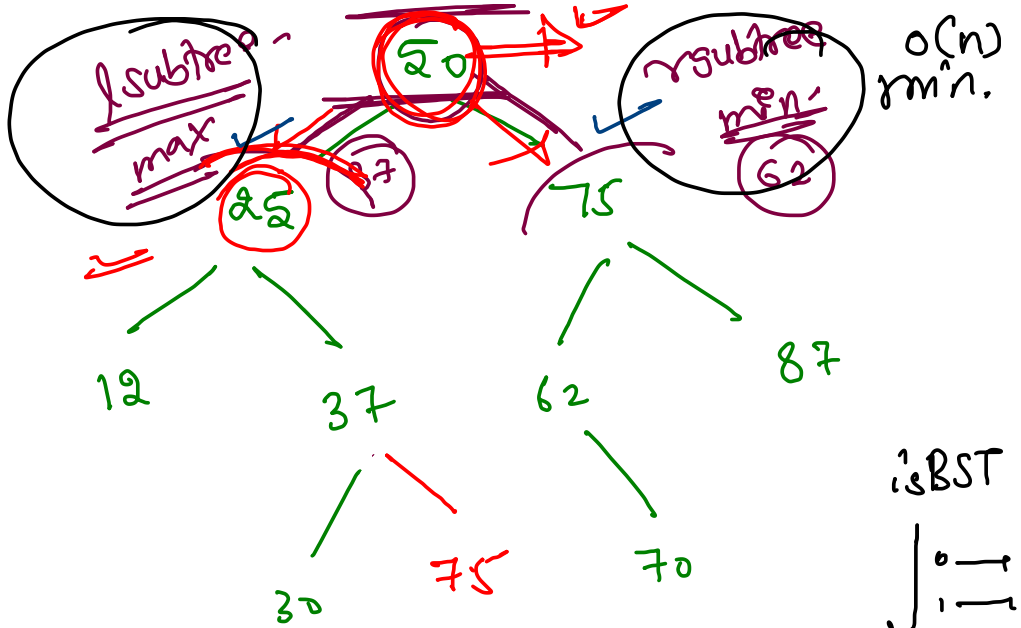
a)- left subtree holds smaller value than node/root.

b)- Right subtree hold greater value than node/root.

c)- Conclusion of 'a' & 'b' :-
Inorder of BST is always in sorted order.







O(n) lmax

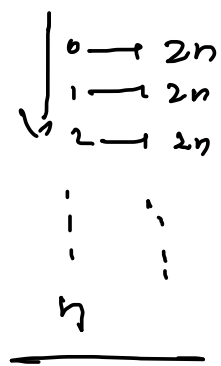
O(n) rmin.

Self check

left check && right

false

isBST



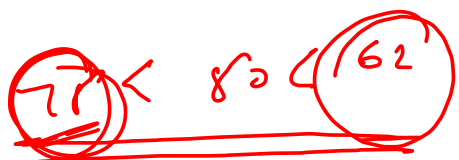
left check && right check

T X

n × 2n : O(n²)

lsubtree max 75

rsubtree min · 62



leftsubtree max < 50 < rightsubtree min

res = res1 || call( )

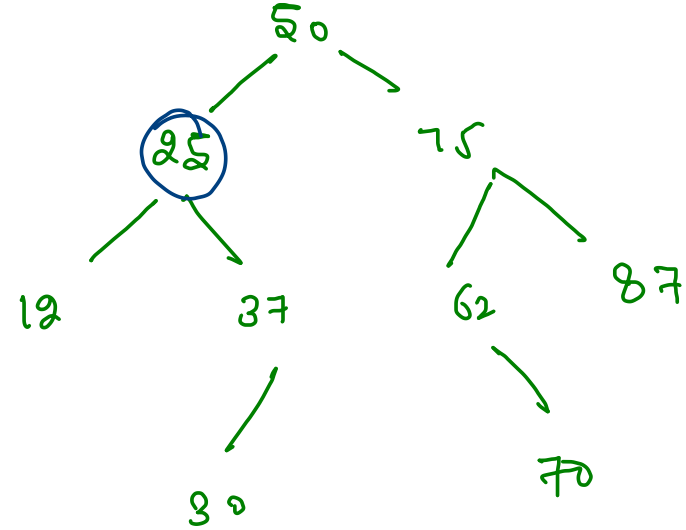
X

T

```java
public static class BSTPair {
    int min;
    int max;
    boolean isBST;

    public BSTPair() {
        this.min = Integer.MAX_VALUE;
        this.max = Integer.MIN_VALUE;
        this.isBST = true;
    }
}
```

wrapper class




```java
public static BSTPair isBST2(Node node) {
    if(node == null) {
        return new BSTPair();
    }

    BSTPair lpair = isBST2(node.left);
    BSTPair rpair = isBST2(node.right);

    boolean res = lpair.max < node.data && node.data < rpair.min;

    BSTPair mpair = new BSTPair();
    mpair.min = Math.min(node.data, Math.min(lpair.min, rpair.min));
    mpair.max = Math.max(node.data, Math.max(lpair.max, rpair.max));
    mpair.isBST = res && lpair.isBST && rpair.isBST;

    return mpair;
}
```

Multisolver Approach.

12 byte + 3 variable

Is Balance

1. BST.

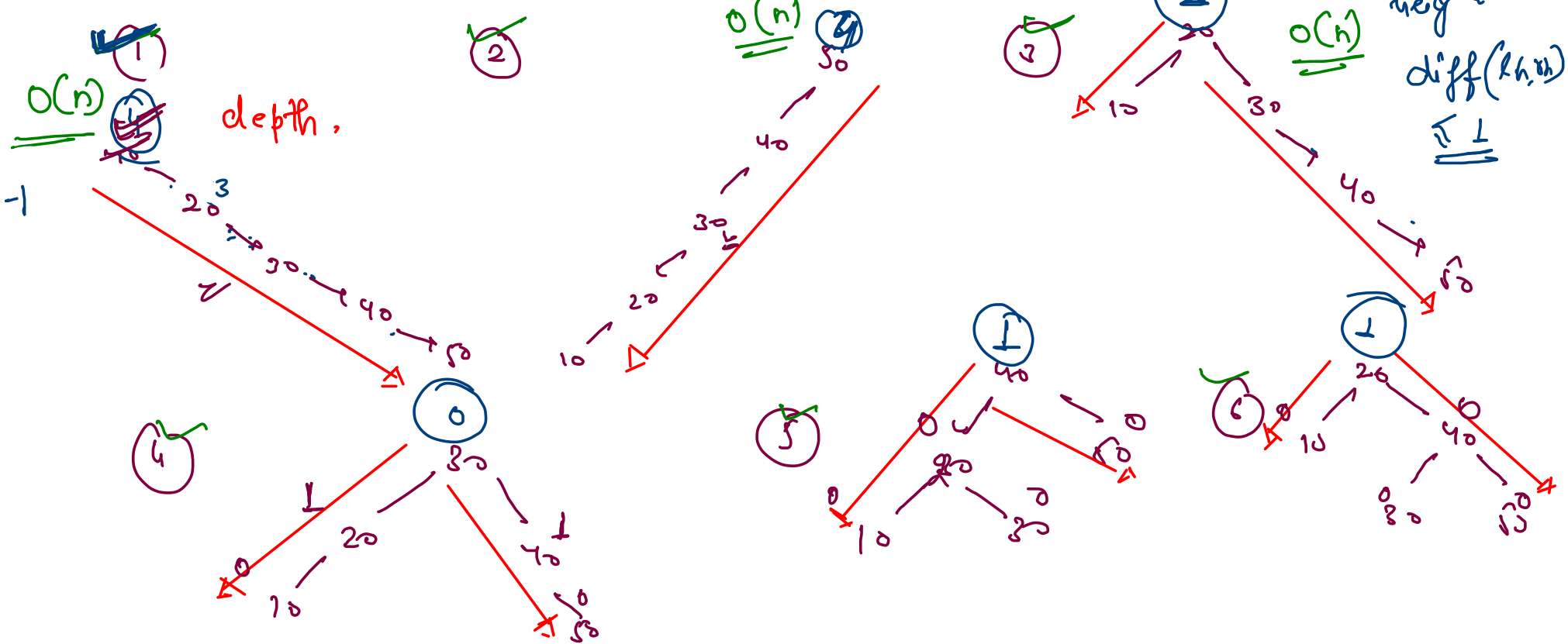
Application → B.S.T. properties

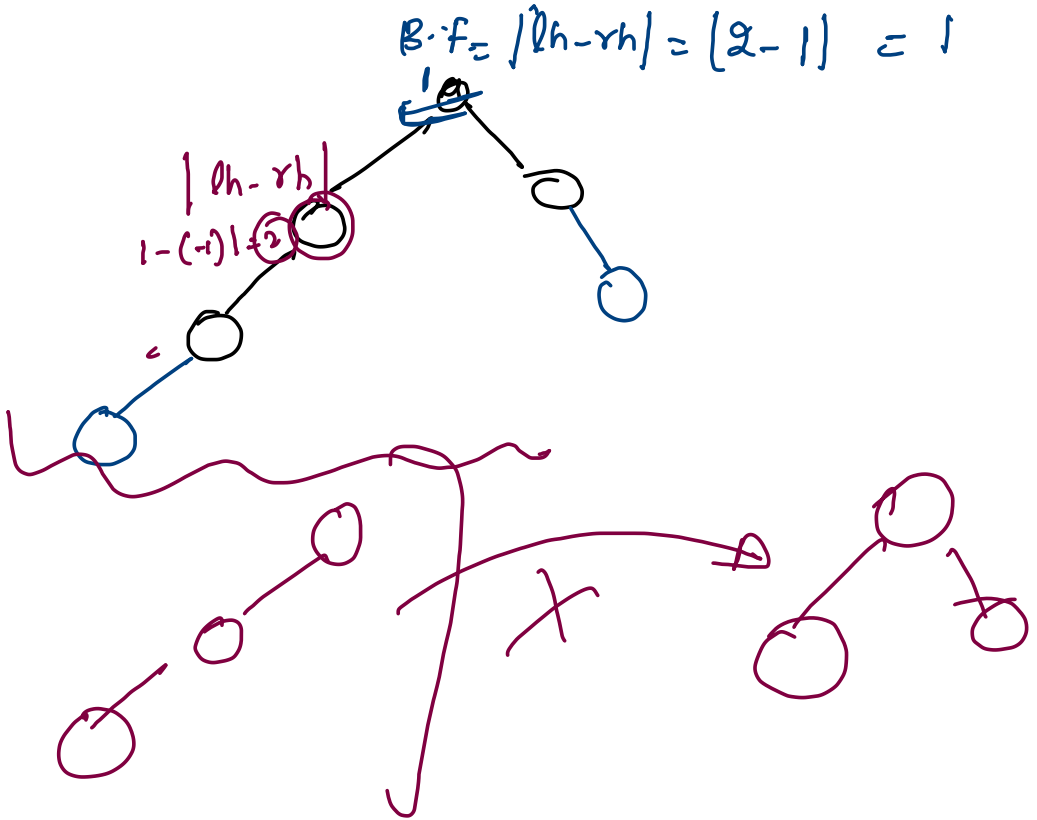
Balancing Factor → Tree

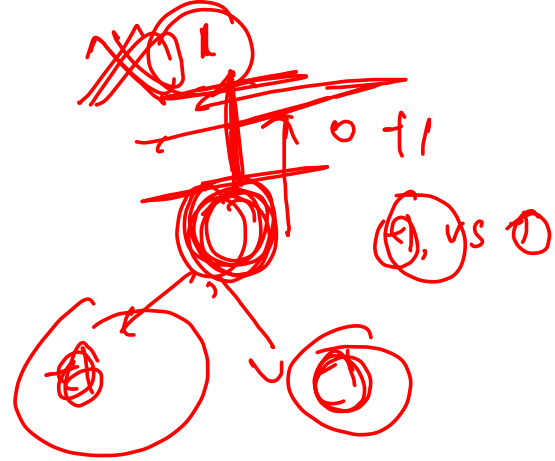
- left height
- right height

diff(lh, rh) ≤ 1

{10, 20, 30, 40, 50}



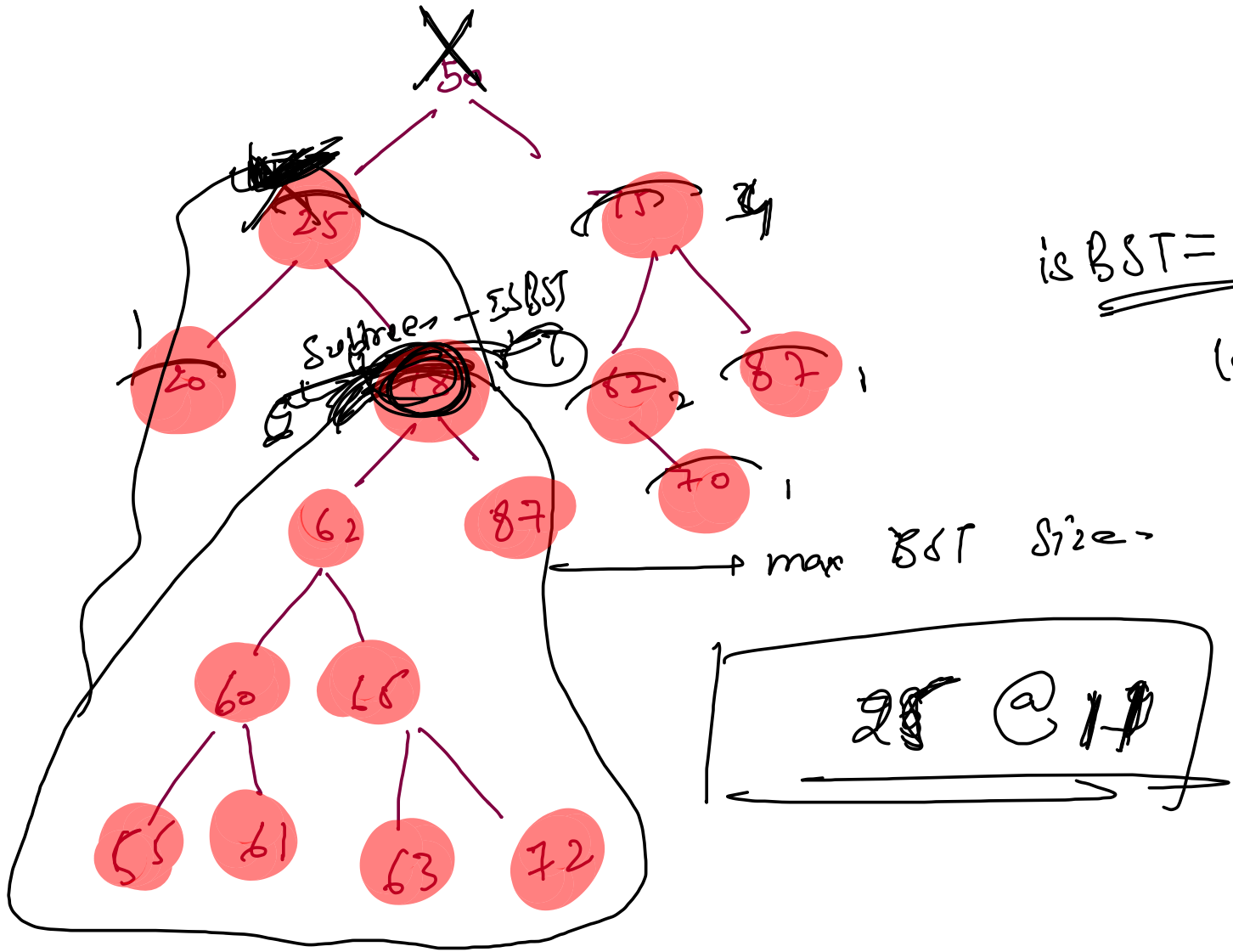




Required to make solution on Every level is ⟶ height

Solution, we are making ⟶ is Balanced.

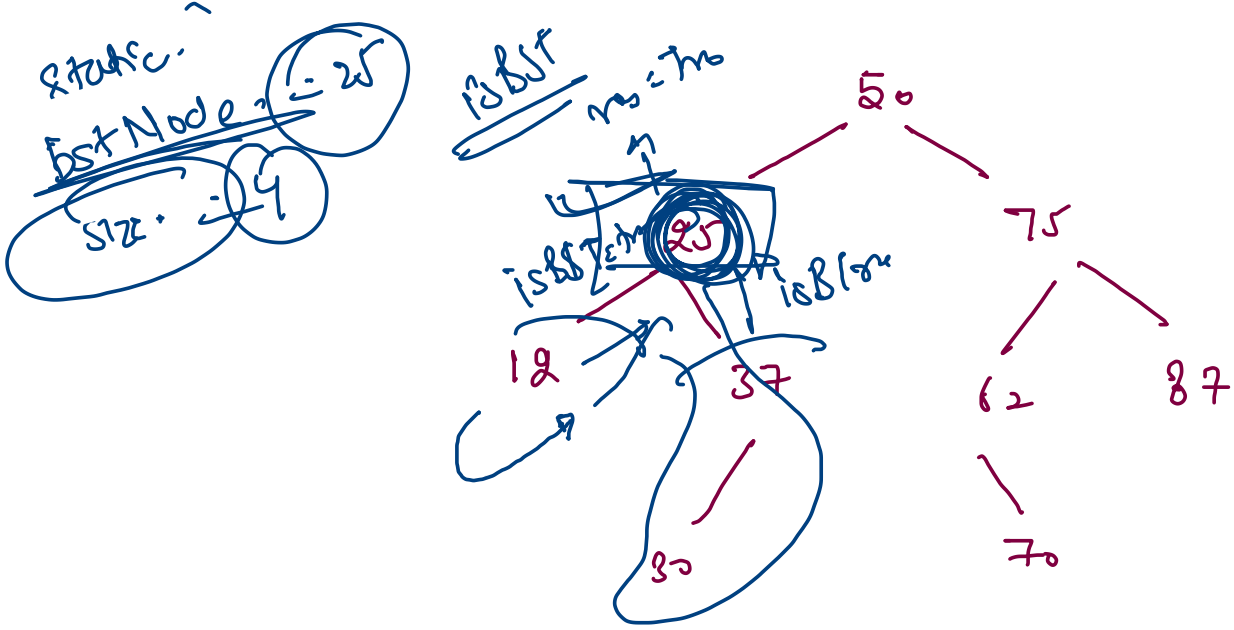
Largest B



is BST = True :

largest subtree

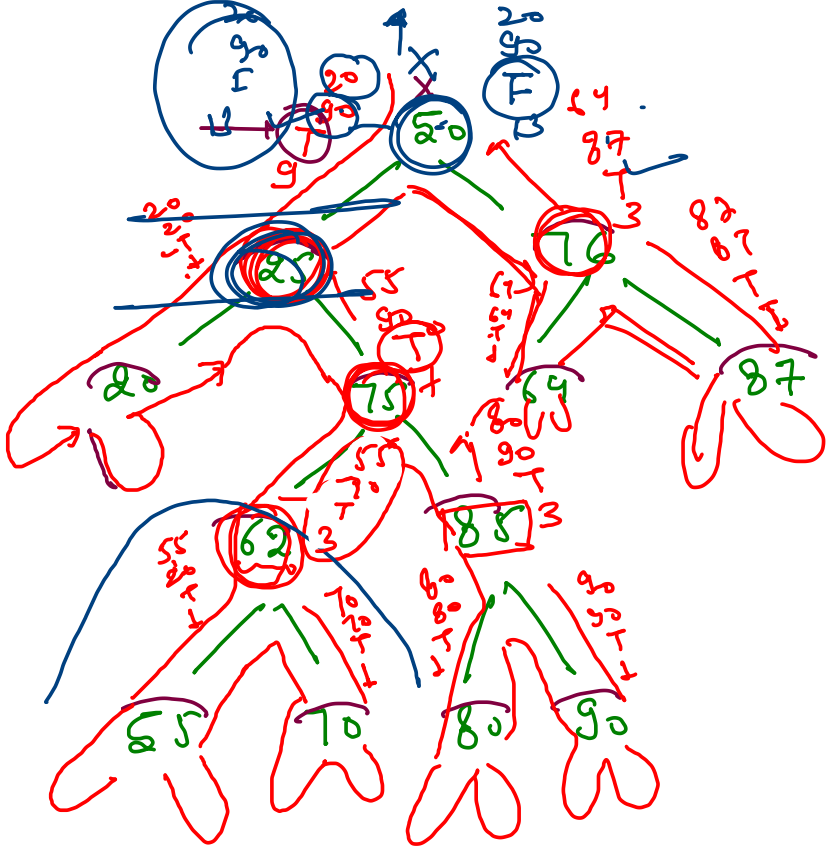
25 @ 11



Post order:

- → isBST
- → min
- → max
- → size.

```
if(size < pair.size){
    size = pair.size.
bstNode = node.
}
```



25 @ 9

BST pair
- min = +∞
- max = −∞
- isBST = True
- size = 0

bstNode = ~~50 62 75 25~~
bstSize = ~~1 3 7~~ 9

```
static Node bstNode = null;
static int bstSize = 0;

public static BSTPair largestBST(Node node) {
    BSTPair lpair = largestBST(node.left);
    BSTPair rpair = largestBST(node.right);

    boolean res = lpair.max < node.data && node.data < rpair.min;

    BSTPair mpair = new BSTPair();
    mpair.min = Math.min(node.data, Math.min(lpair.min, rpair.min));
    mpair.max = Math.max(node.data, Math.max(lpair.max, rpair.max));
    mpair.isBST = res && lpair.isBST && rpair.isBST;
    mpair.size = lpair.size + rpair.size + 1;

    if(mpair.isBST == true && mpair.size > bstSize) {
        bstSize = mpair.size;
        bstNode = node;
    }

    return mpair;
}
```