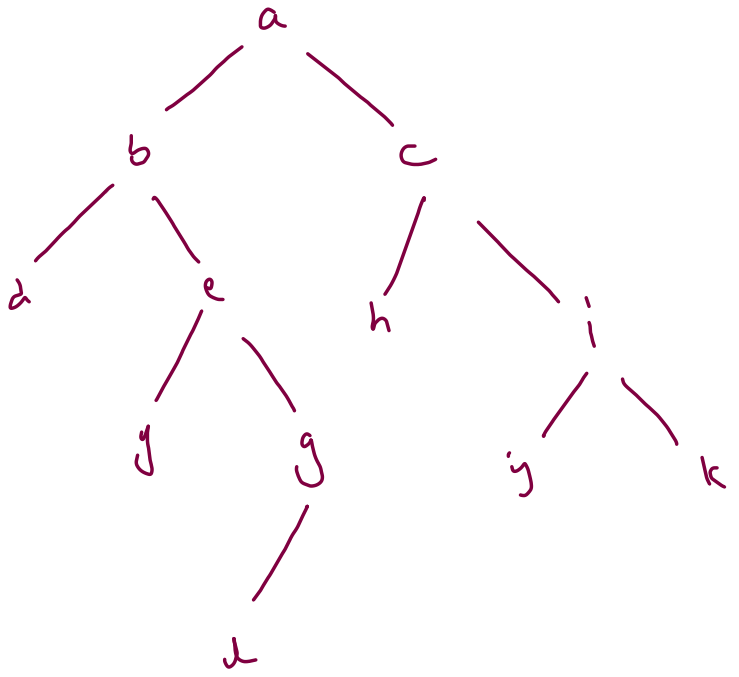
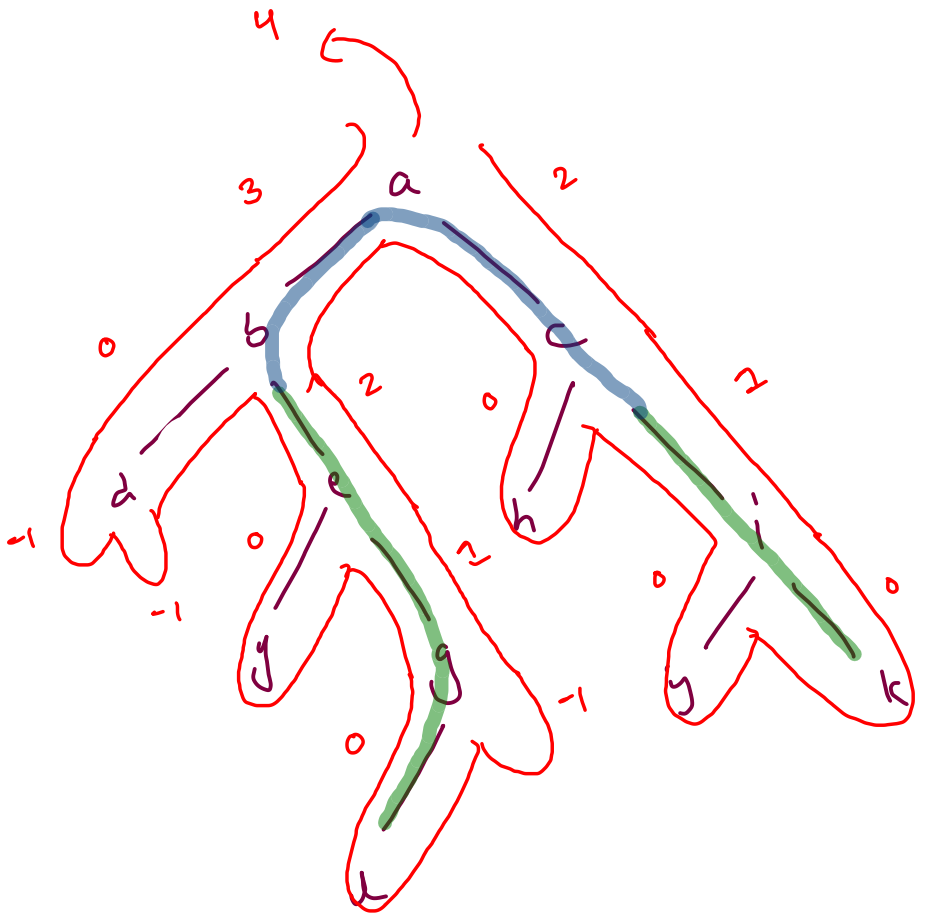
diameter



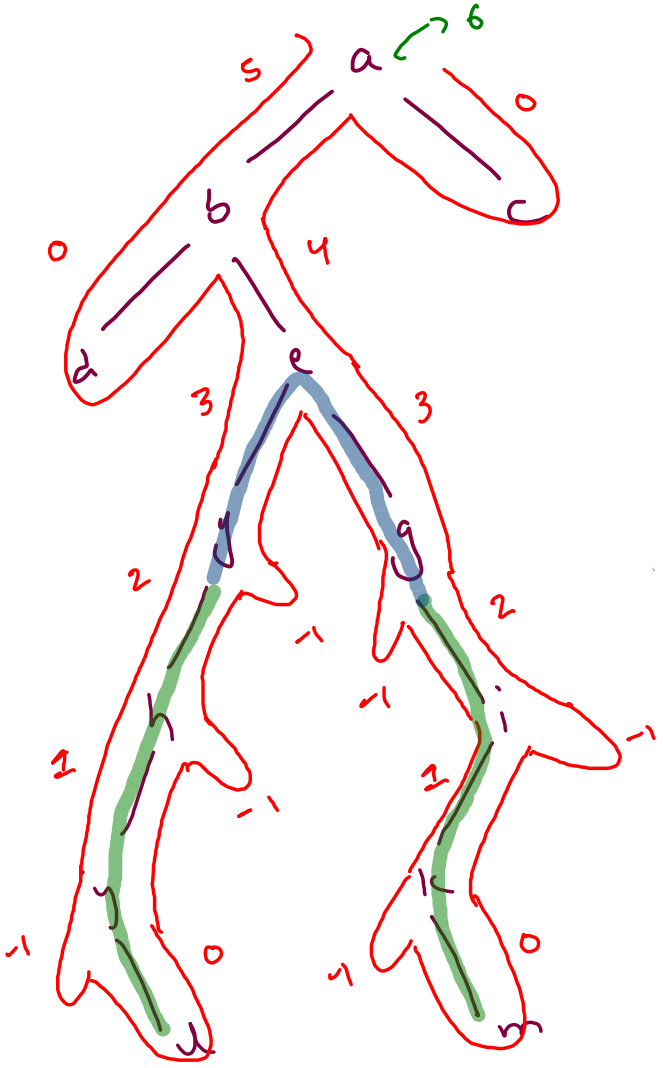
dist = lch + rch + 2

diameter = max (dist)



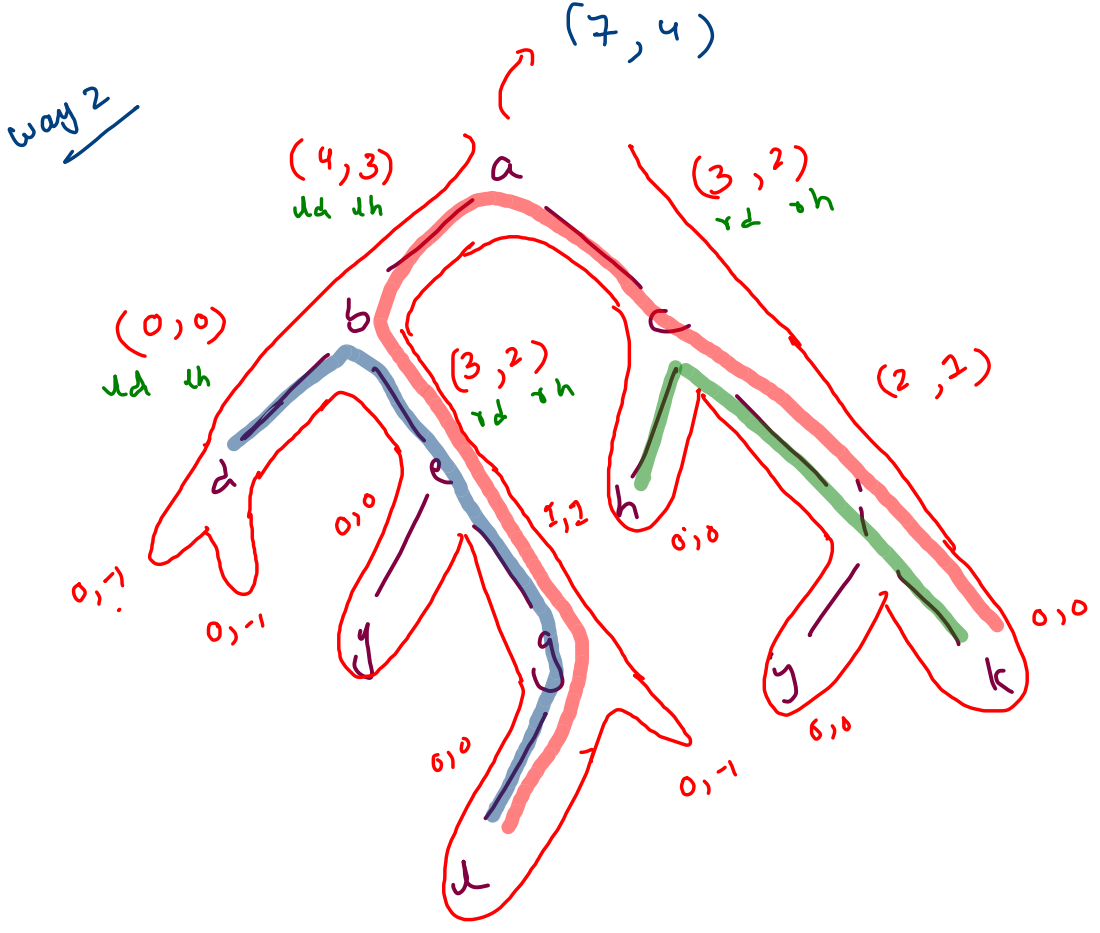
dia = ~~0~~ ~~1~~ ~~3~~ 4 7

```
public static int helper(Node node) {
    if(node == null) {
        return -1;
    }

    int lch = helper(node.left);
    int rch = helper(node.right);

    int dist = lch + rch + 2;

    if(dist > dia) {
        dia = dist;
    }

    int ht = Math.max(lch,rch) + 1;

    return ht;

}
```



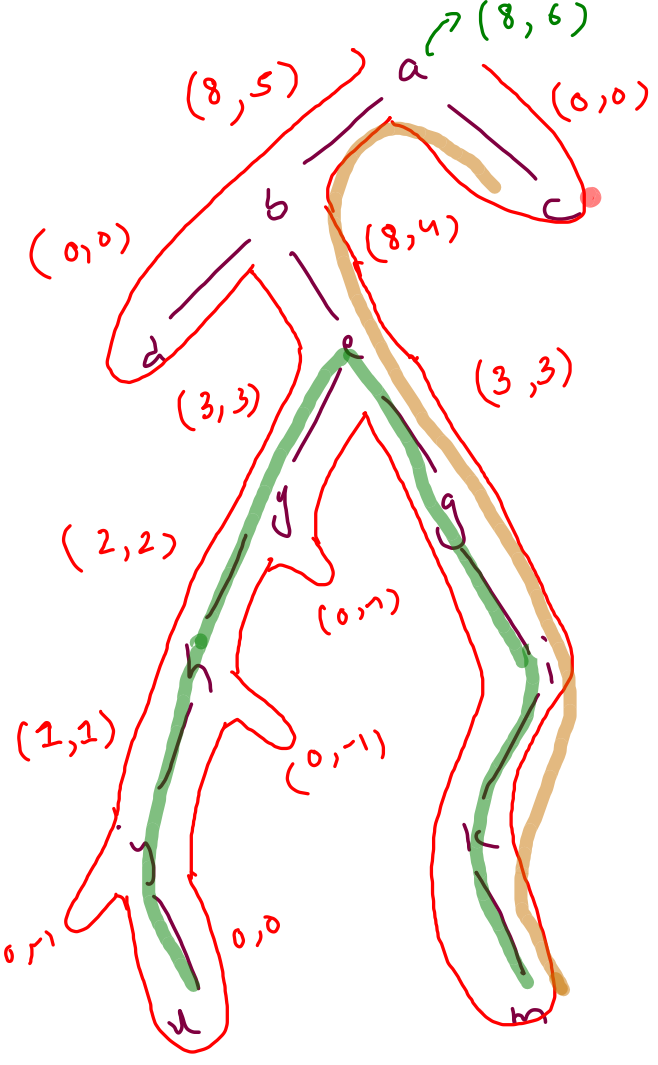
dia = ~~9~~ ~~7~~
~~7~~
~~8~~
8

```java
public static int helper(Node node) {
        if(node == null) {
            return -1;
        }

        int lch = helper(node.left);
        int rch = helper(node.right);

        int dist = lch + rch + 2;

        if(dist > dia) {
            dia = dist;
        }

        int ht = Math.max(lch,rch) + 1;

        return ht;
}
```



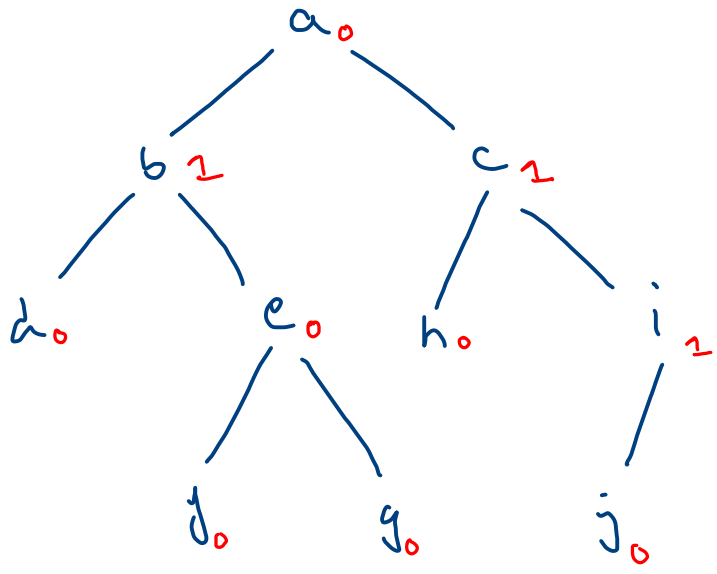
(dia, ht)

```java
public static DiaPair helper(Node node) {
  if(node == null) {
      return new DiaPair(0,-1);
  }

  DiaPair lp = helper(node.left);
  DiaPair rp = helper(node.right);

  int dist = lp.ht + rp.ht + 2;
  int dia = Math.max(Math.max(lp.dia,rp.dia),dist);
  int ht = Math.max(lp.ht,rp.ht) + 1;

  return new DiaPair(dia,ht);
}
```




```java
public static DiaPair helper(Node node) {
  if(node == null) {
      return new DiaPair(0,-1);
  }

  DiaPair lp = helper(node.left);
  DiaPair rp = helper(node.right);

  int dist = lp.ht + rp.ht + 2;
  int dia = Math.max(Math.max(lp.dia,rp.dia),dist);
  int ht = Math.max(lp.ht,rp.ht) + 1;

  return new DiaPair(dia,ht);
}
```

is balanced tree



balanced

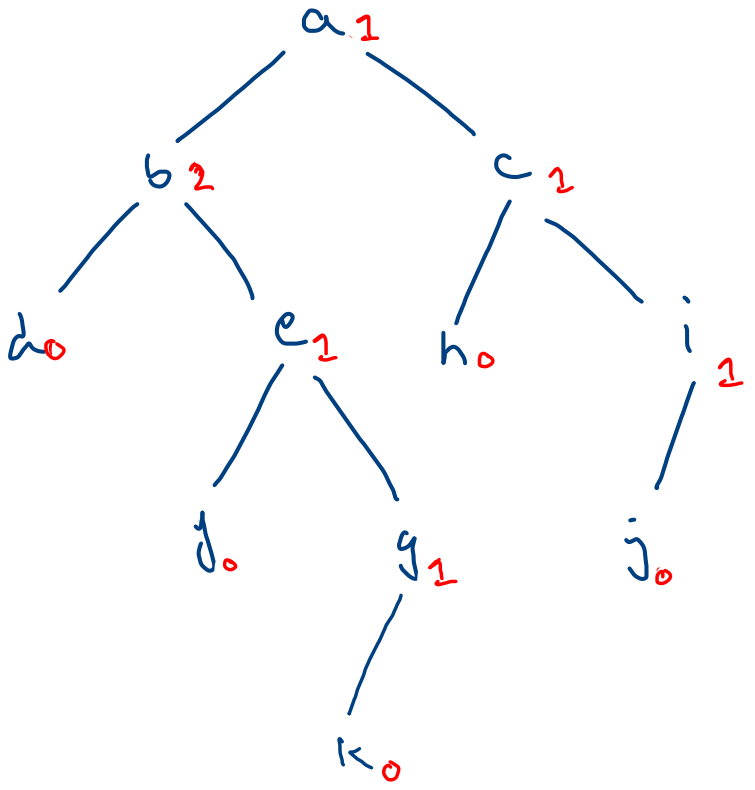
(1) balacing factor

bf = |lh - rh|

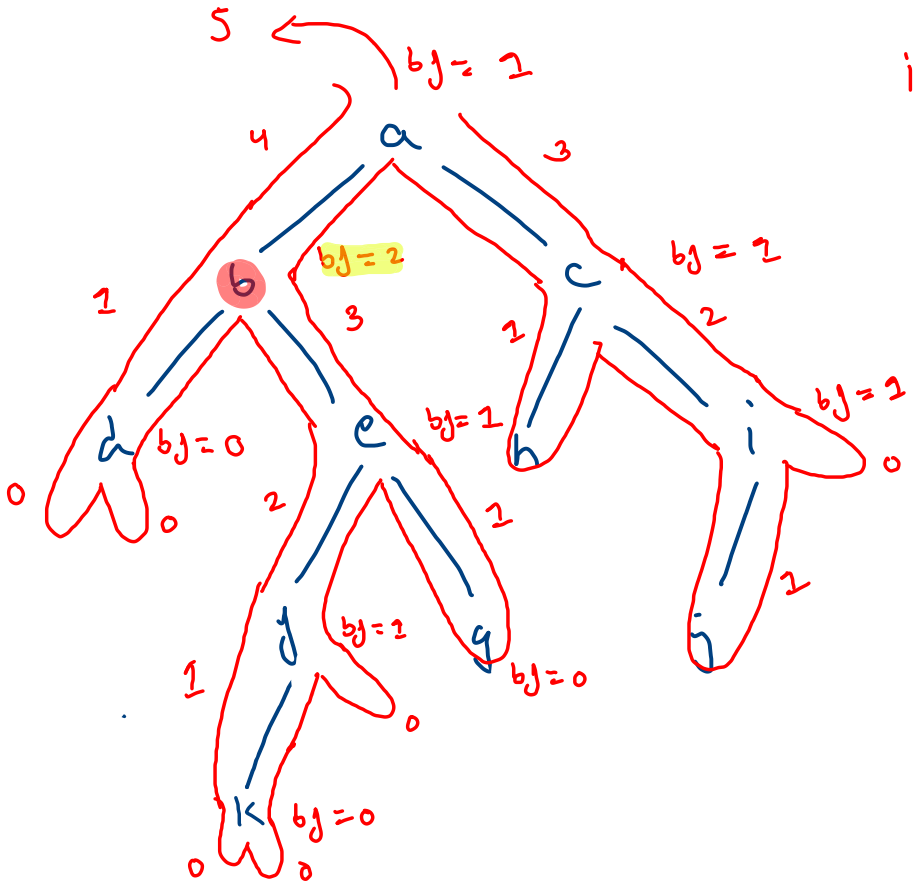
(height(node))

(2) node is balanced when it $\boxed{bf <= 1}$

(3) A tree is balanced when all nodes are balanced

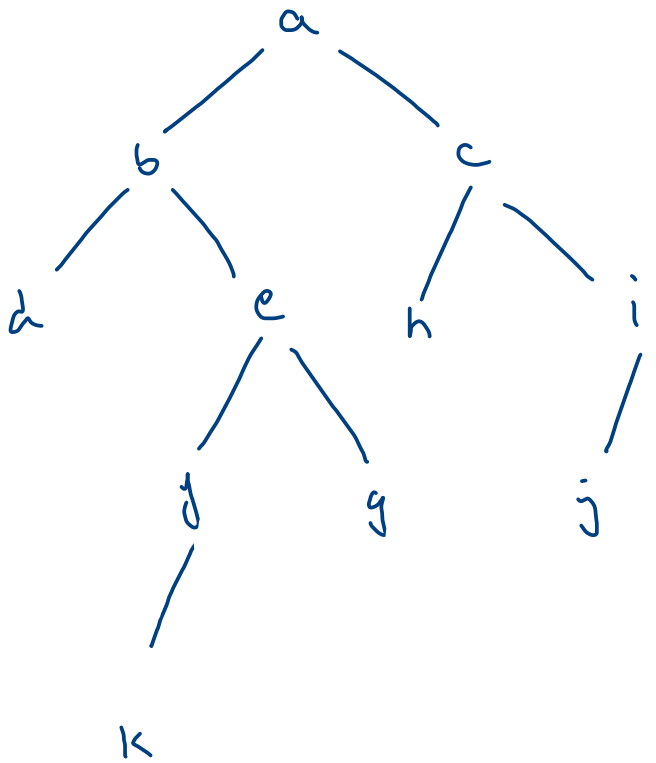


not balanced (due to b)



isBal = ~~F~~

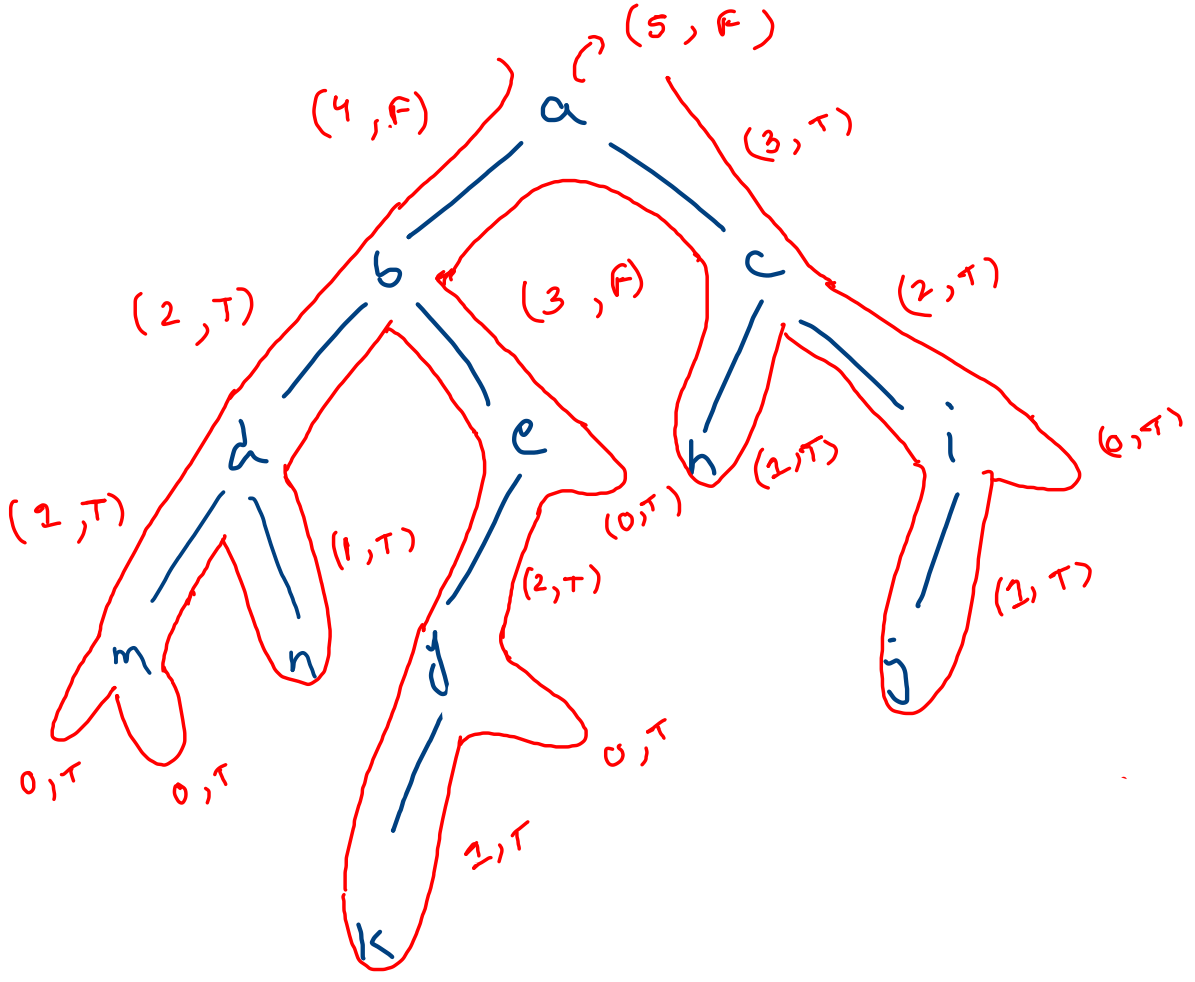
```
public static int helper(Node node) {
  if(node == null) {
      return 0; //height in terms of node
  }

  int lh = helper(node.left);
  int rh = helper(node.right);

  int bf = Math.abs(lh - rh); //balancing factor

  if(bf > 1) {
      isBal = false;
  }

  return Math.max(lh,rh) + 1;
}
```




```
boolean isBalanced(root){
    if(root == null){
        return;
    }
    boolean la = isBalanced(root.left);
    boolean ra = isBalanced(root.right);
    int bf = |height(root.left) -
             height(root.right)|;
    boolean isBal = la && ra &&
                    (bf <= 1);
    return;
}
```

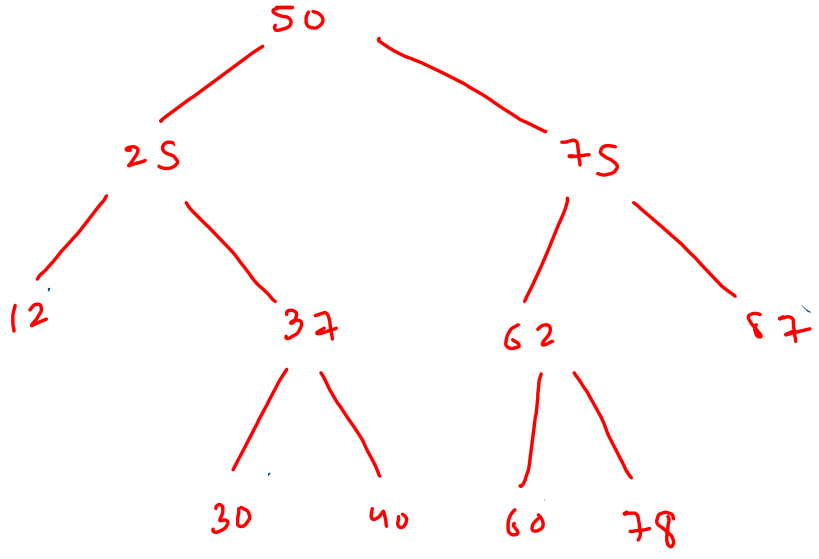
$O(n^2)$

X




```java
public static BalPair helper(Node node) {
  if(node == null) {
      return new BalPair(0,true);
  }

  BalPair lp = helper(node.left);
  BalPair rp = helper(node.right);

  int bf = Math.abs(lp.ht-rp.ht); //balancing factor
  boolean isBal = lp.isBal && rp.isBal && (bf <= 1);
  int ht = Math.max(lp.ht,rp.ht) + 1;

  return new BalPair(ht,isBal);
}
```

Binary search tree



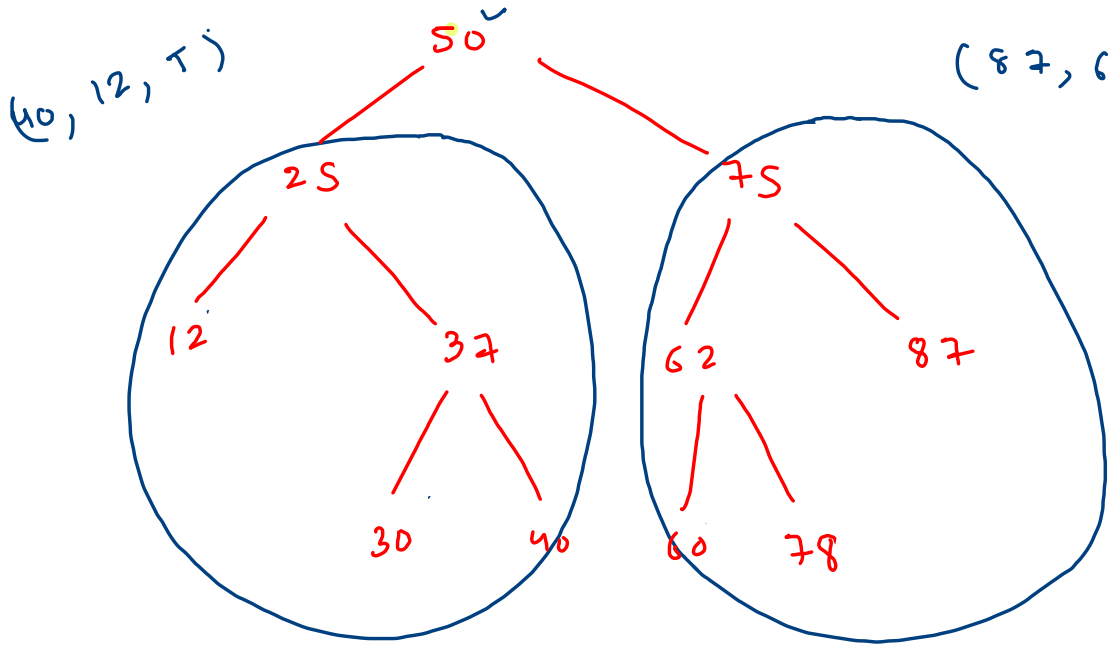
prop

∀ nodes in left subtree < node.val < ∀ nodes in right subtree

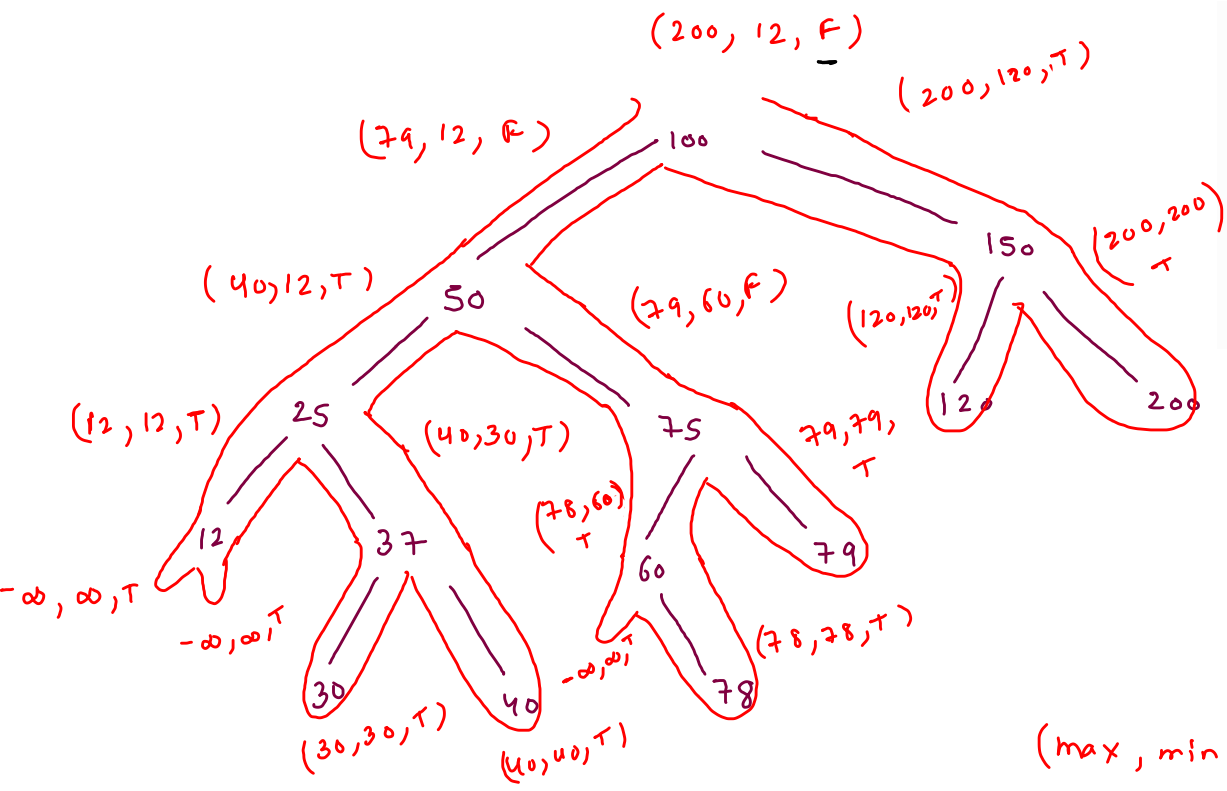
when BST ?

-> all nodes in the binary tree follows the above prop.

(87, 12, F)

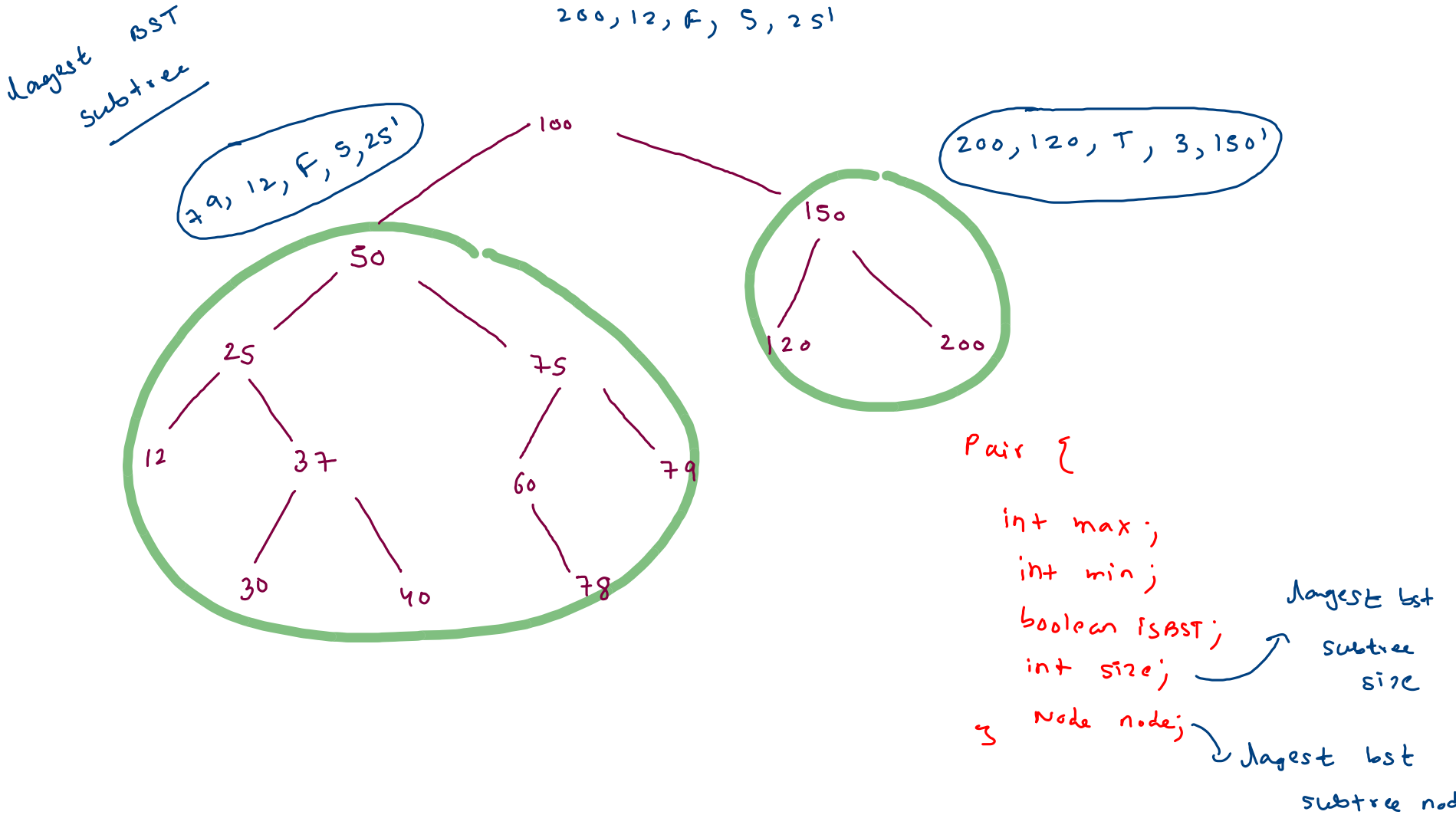



```
Pair {
    int max;
    int min;
    boolean isBST;
}
```

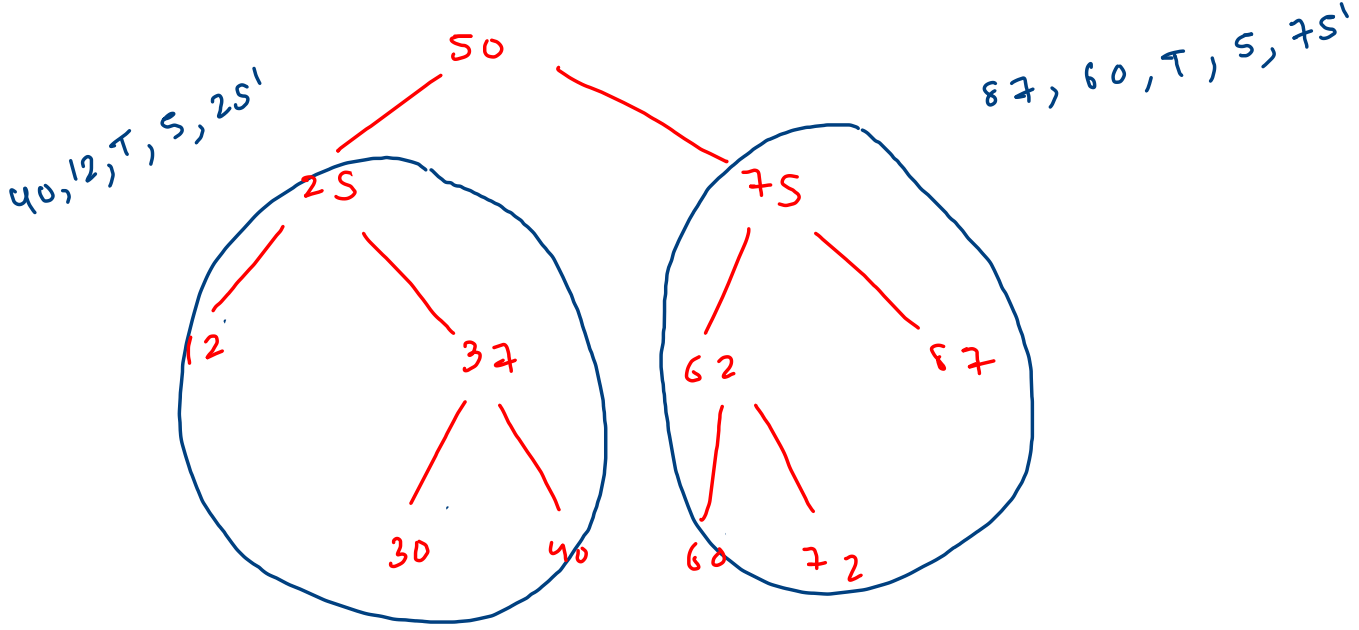



```java
public static BSTPair helper(Node node) {
  if(node  == null) {
      return new BSTPair(Integer.MIN_VALUE,Integer.MAX_VALUE,true);
  }

  BSTPair lp = helper(node.left);
  BSTPair rp = helper(node.right);

  int max = Math.max(Math.max(lp.max,rp.max),node.data);
  int min = Math.min(Math.min(lp.min,rp.min),node.data);

  boolean isBST = lp.isBST && rp.isBST && (lp.max < node.data && rp.min > node.data);

  return new BSTPair(max,min,isBST);

}
```

(max, min, isBST)


largest BST subtree
200, 12, F, 5, 25'
100
79, 12, F, 5, 25'
200, 120, T, 3, 150'
50
150
25
75
120
200
12
37
60
79
30
40
78
Pair {
int max;
int min;
boolean isBST;
int size;
largest bst subtree size
Node node;
largest bst subtree node
}

87, 12, T, 11, 50'



```java
int max = Math.max(Math.max(lp.max,rp.max),node.data);
int min = Math.min(Math.min(lp.min,rp.min),node.data);
boolean isBST = lp.isBST && rp.isBST && (lp.max < node.data && rp.min > node.data);

int size = 0;
Node lbn = null;

if(isBST == true) {
    size = lp.size + rp.size + 1;
    lbn = node;
}
else {
    if(lp.size > rp.size) {
        size = lp.size;
        lbn = lp.node;
    }
    else {
        size = rp.size;
        lbn = rp.node;
    }
}
```