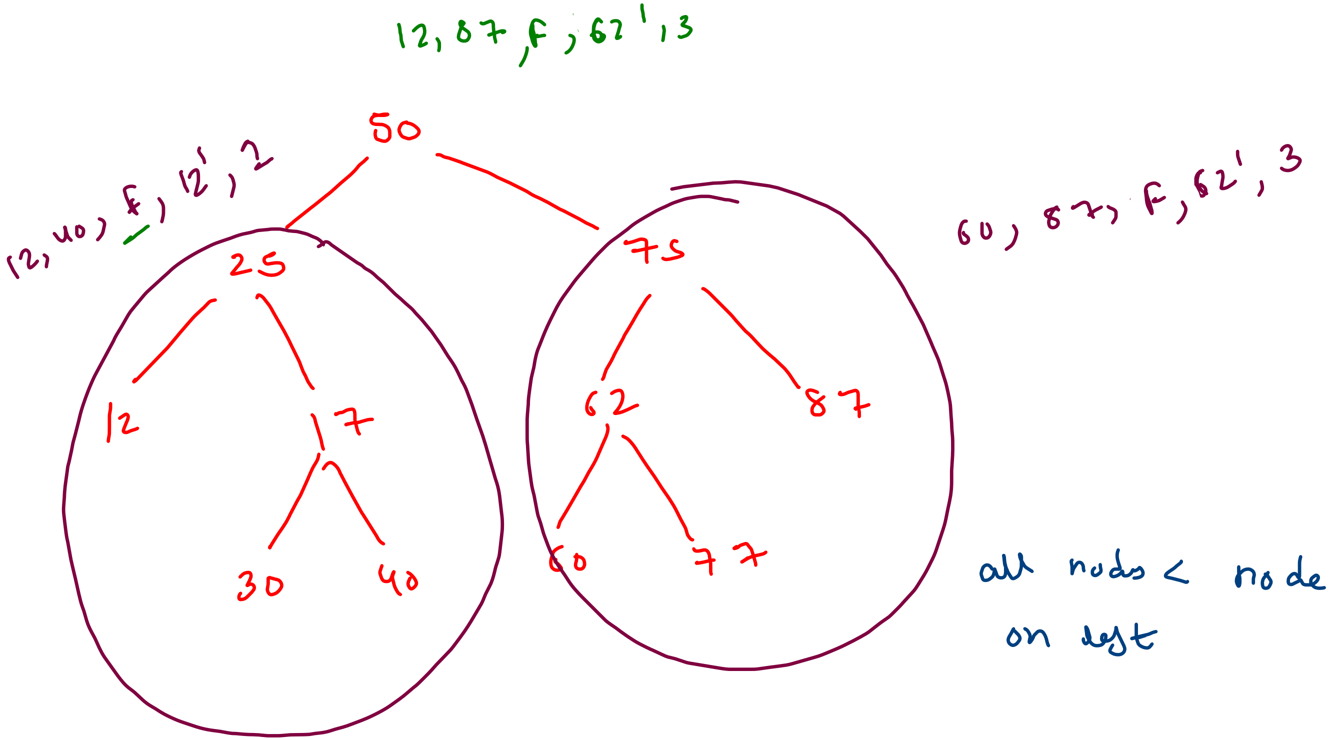
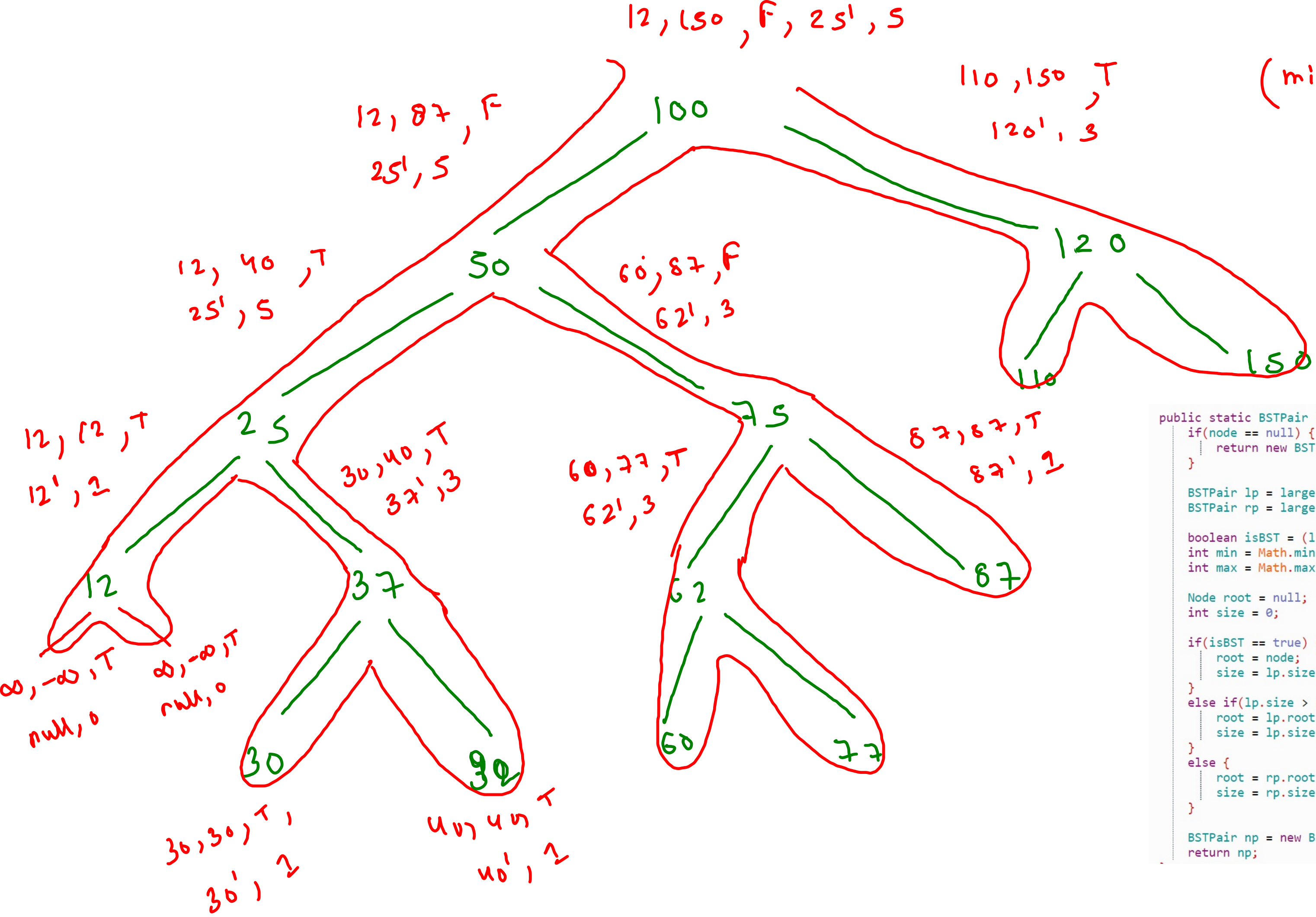
largest bst subtree



all nodes < node.data < all nodes
on left on right

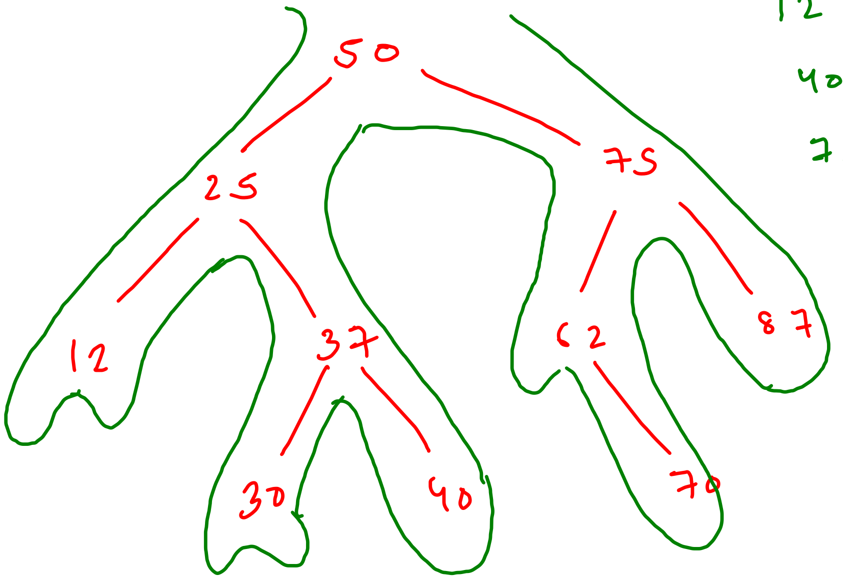
(min, max, isBST, root, size)



```java
public static BSTPair largestBSTSubtree(Node node) {
    if(node == null) {
        return new BSTPair(Integer.MAX_VALUE,Integer.MIN_VALUE,true,0,null);
    }

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

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

    Node root = null;
    int size = 0;

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

    BSTPair np = new BSTPair(min,max,isBST,size,root);
    return np;
```

BST

Binary search tree



12 25 30 37

40 50 62 70

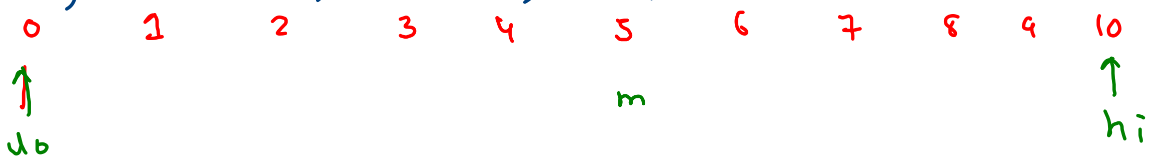
75 87

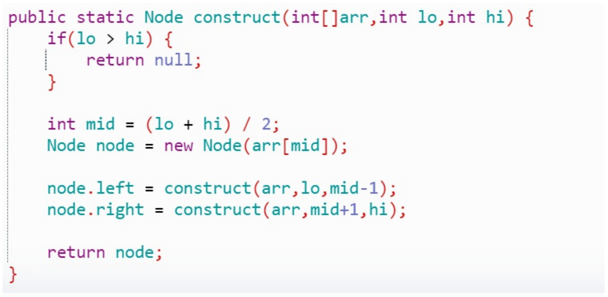
Inorder -> sorted

left node right

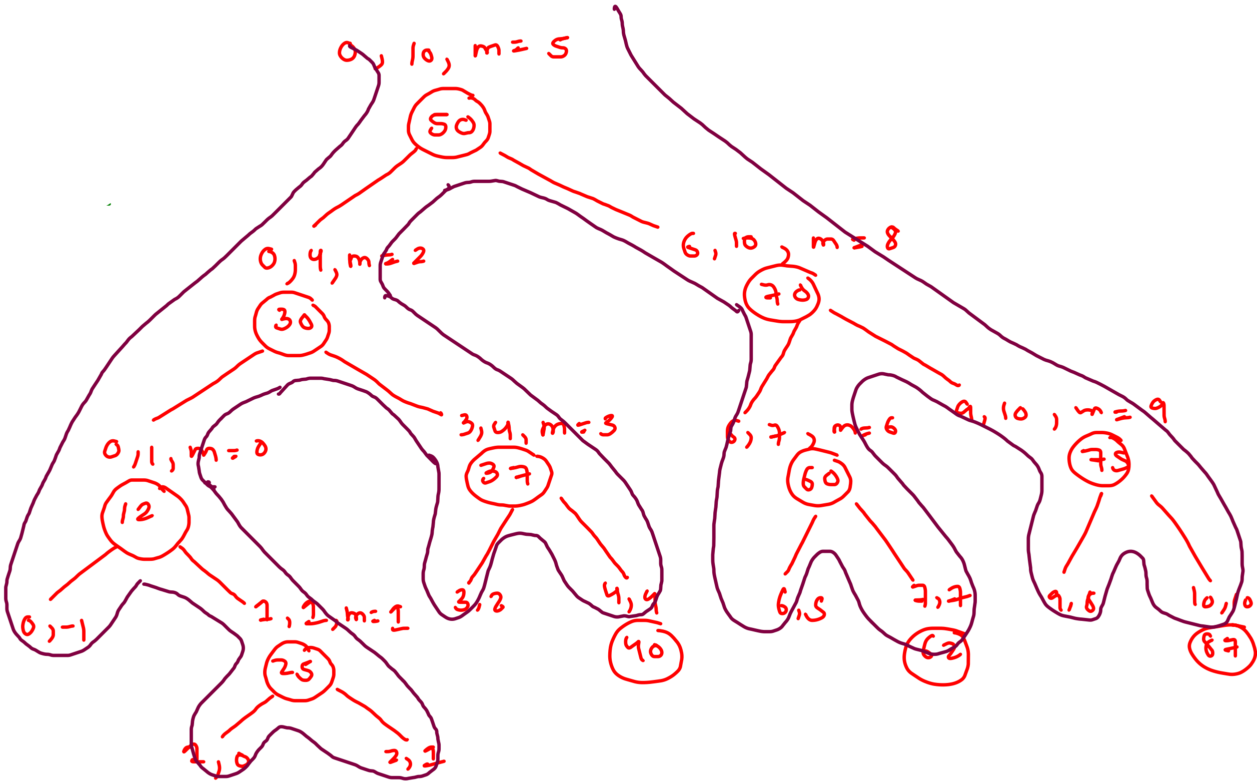
all nodes < node-data < all nodes
on left on right

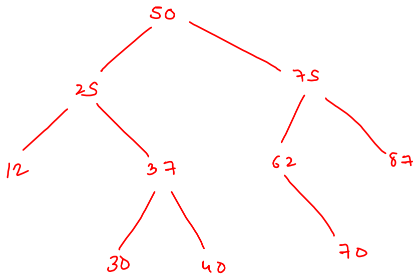
arr = [12, 25, 30, 37, 40, 50, 60, 62, 70, 75, 87]



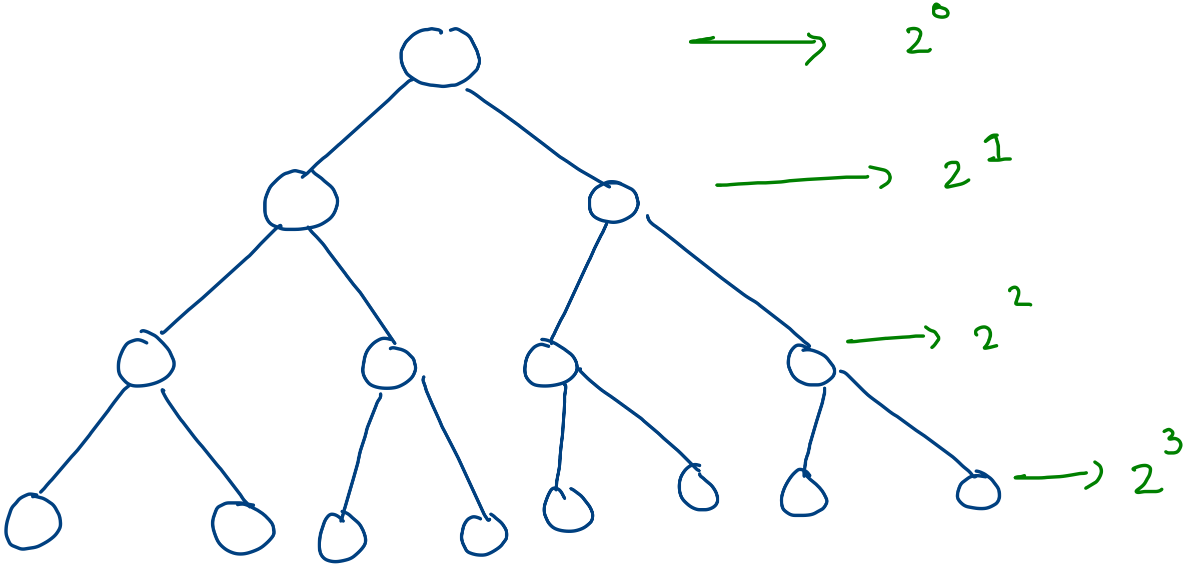


```
public static Node construct(int[]arr,int lo,int hi) {
    if(lo > hi) {
        return null;
    }

    int mid = (lo + hi) / 2;
    Node node = new Node(arr[mid]);

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

    return node;
}
```





$h = \log n$



2^{da}

$$n = 2^0 + 2^1 + 2^2 + \dots + 2^{h-1}$$

$a = 1$

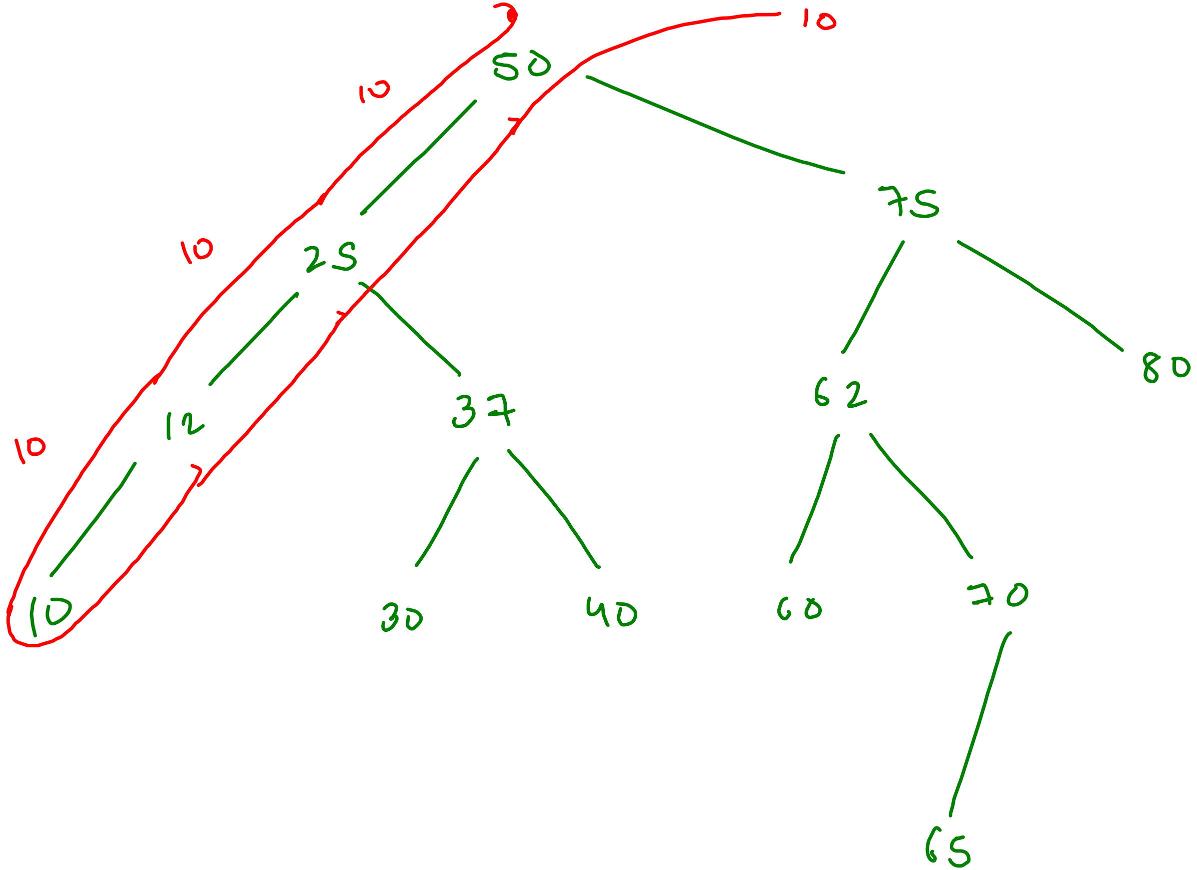
$r = 2$

$$sum = \frac{ar^{t-1}}{r-1}$$

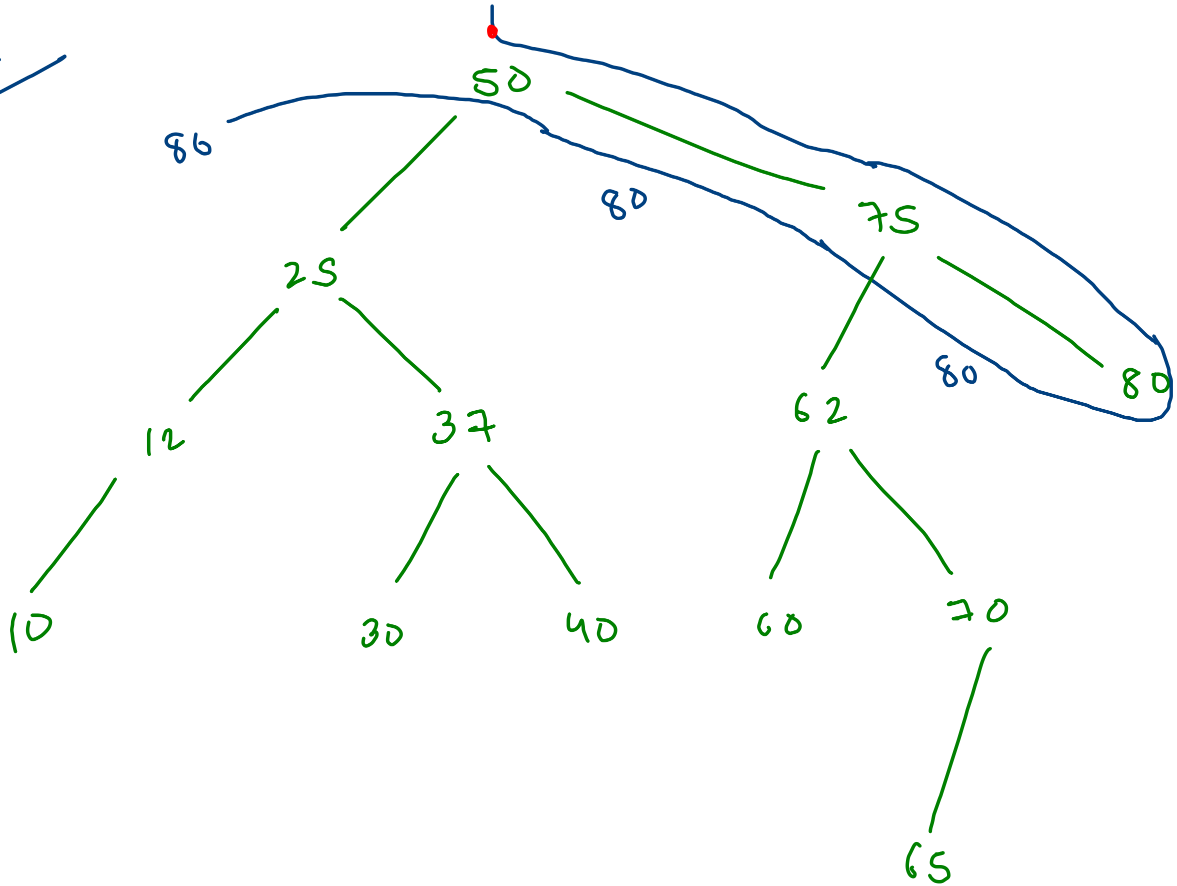
$$n = 1 \times 2^h$$

$$\boxed{\log_2 n = h}$$

minimum

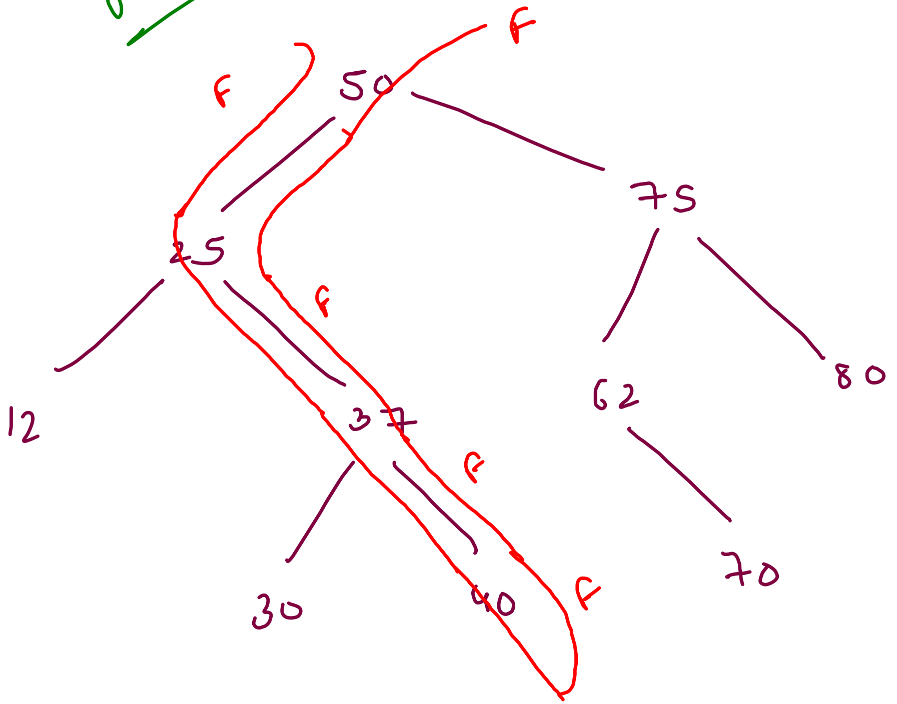


maximum

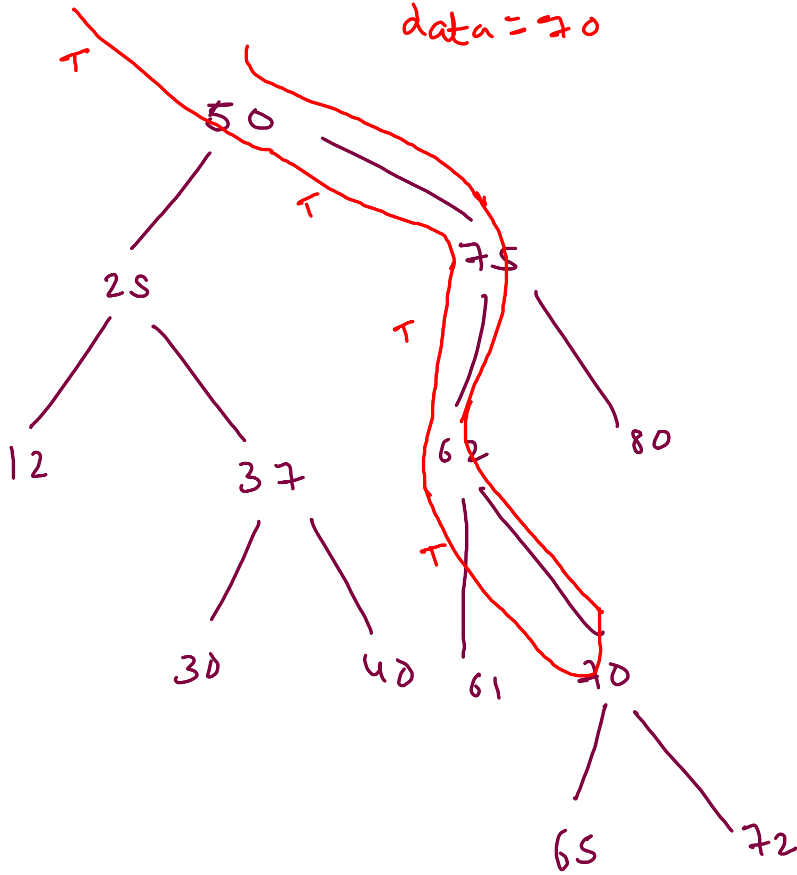


find

data = 45

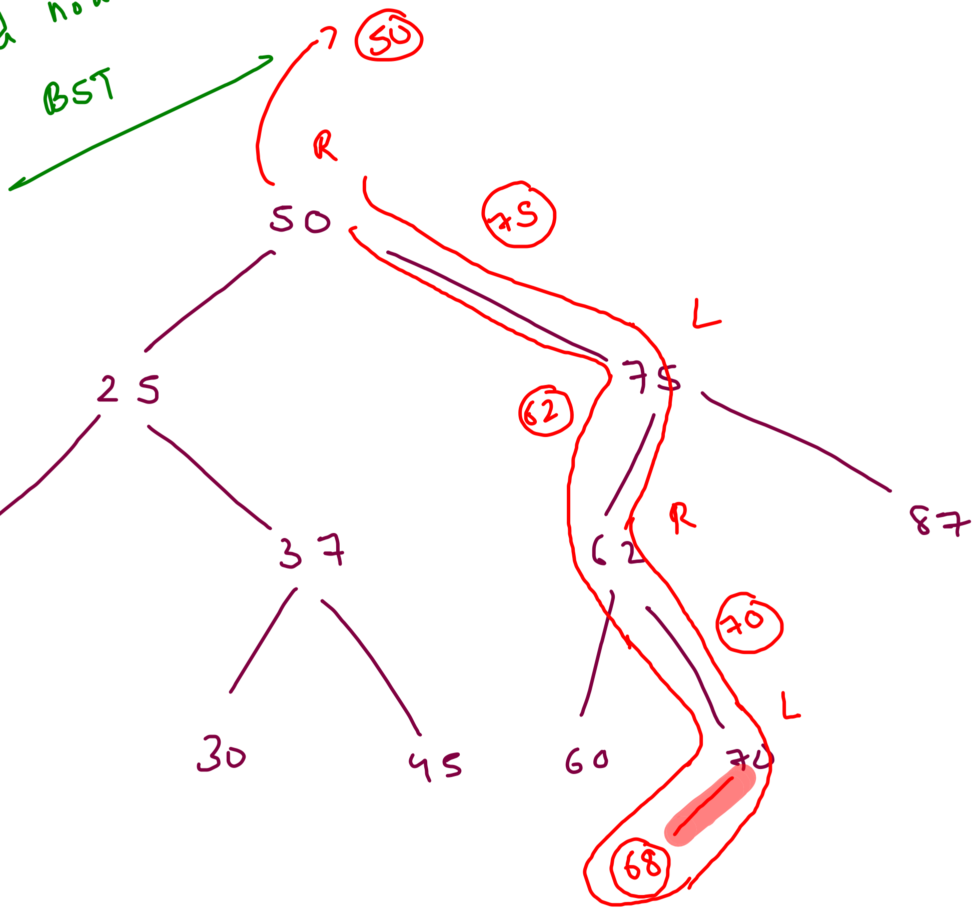


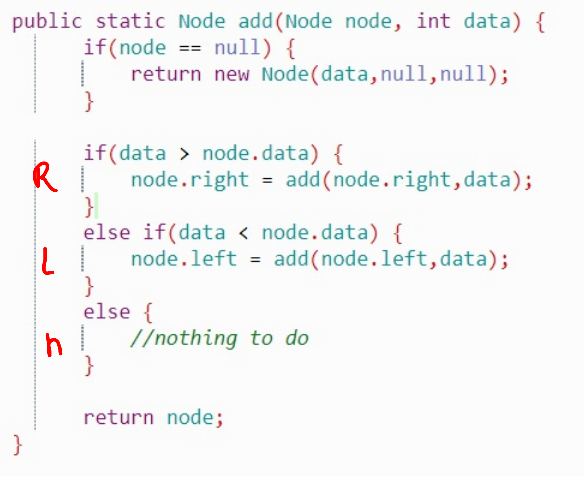
data = 70



Add node in BST

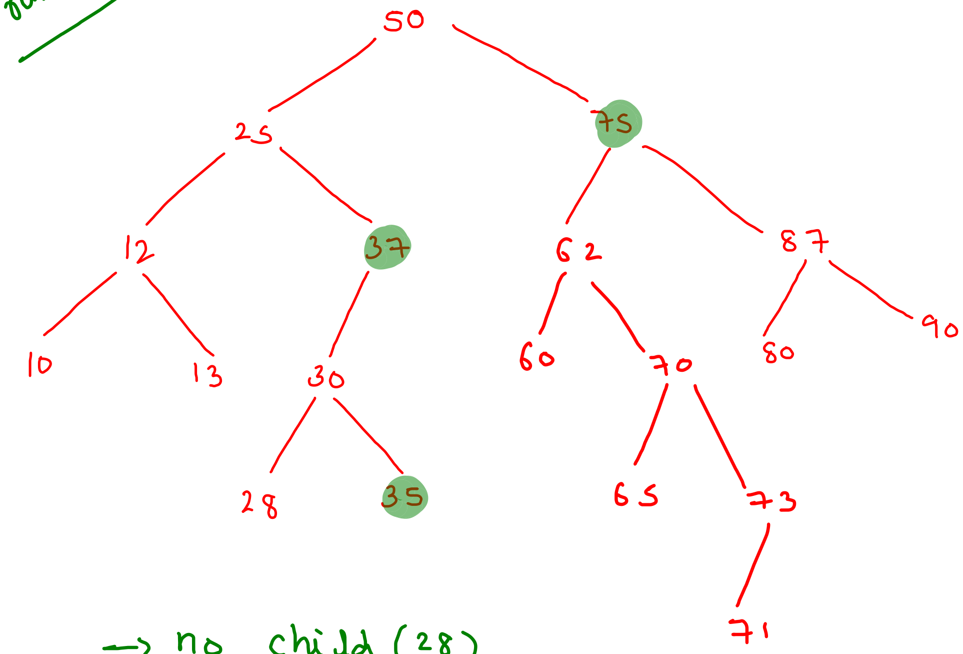
data = 68





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

    if(data > node.data) {
        node.right = add(node.right,data);
    }
    else if(data < node.data) {
        node.left = add(node.left,data);
    }
    else {
        //nothing to do
    }

    return node;
}
```

remove



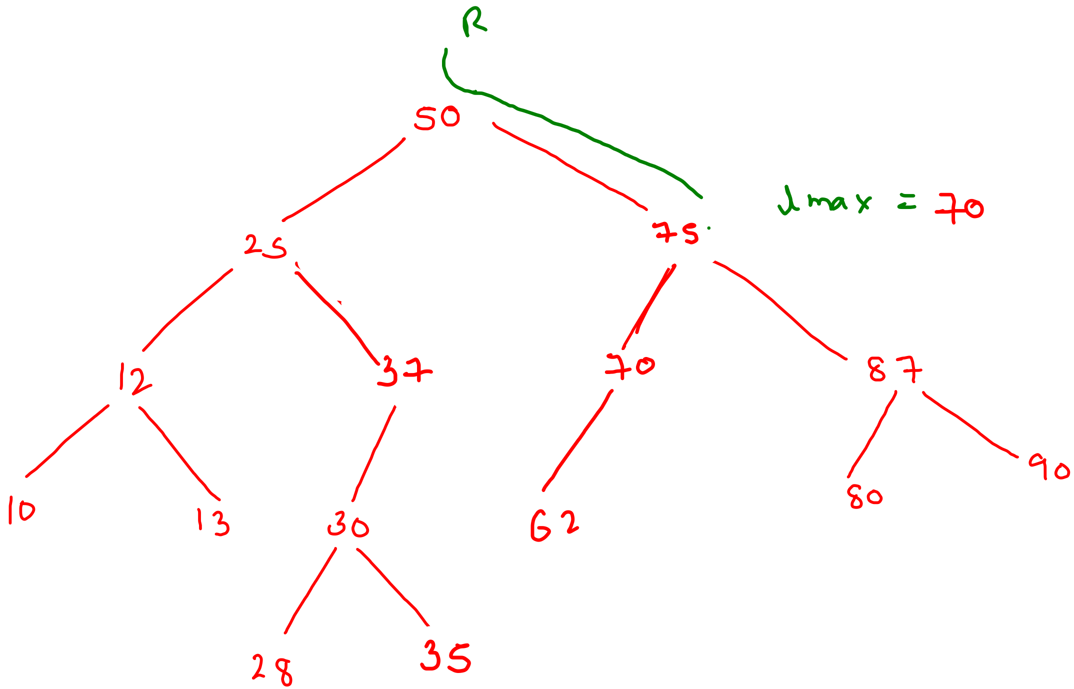
→ no child (28)

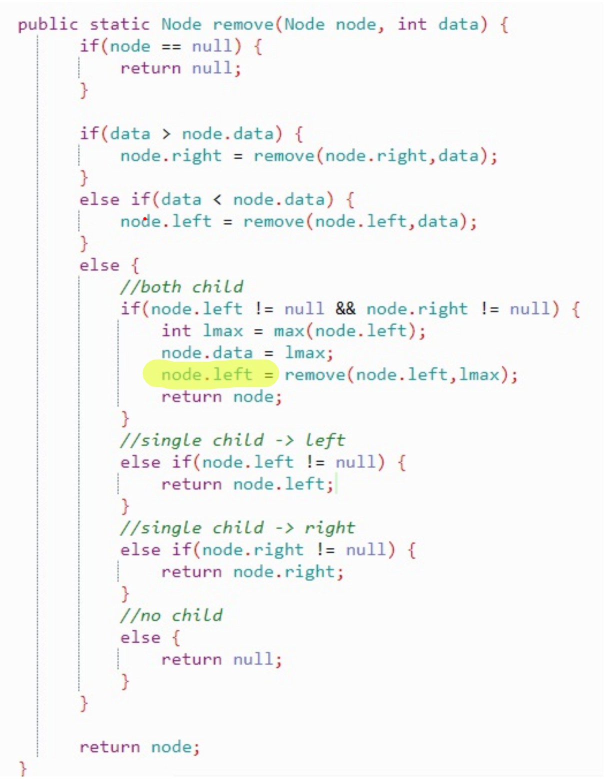
→ single child (37)

→ both child (75)

```
if (no child) {
        return null;
}
else if (single child) {
        return single child;
}
else if ( both child ) {
        lmax = max (node.left);
        node.data = lmax;
        node.left = remove (node.left, lmax);
}
```

data = 75




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

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

    return node;
}
```