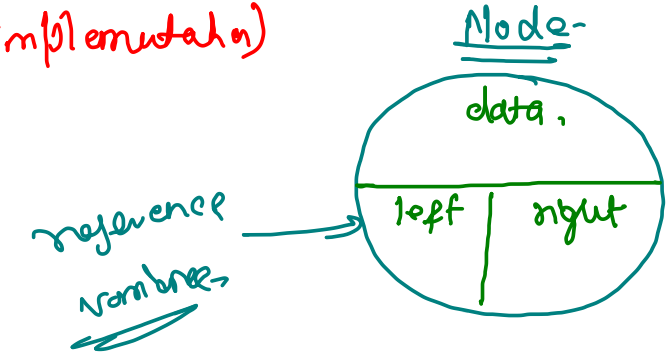
Binary Tree

Infix Evaluation → Application

(Structured Based Implementation)

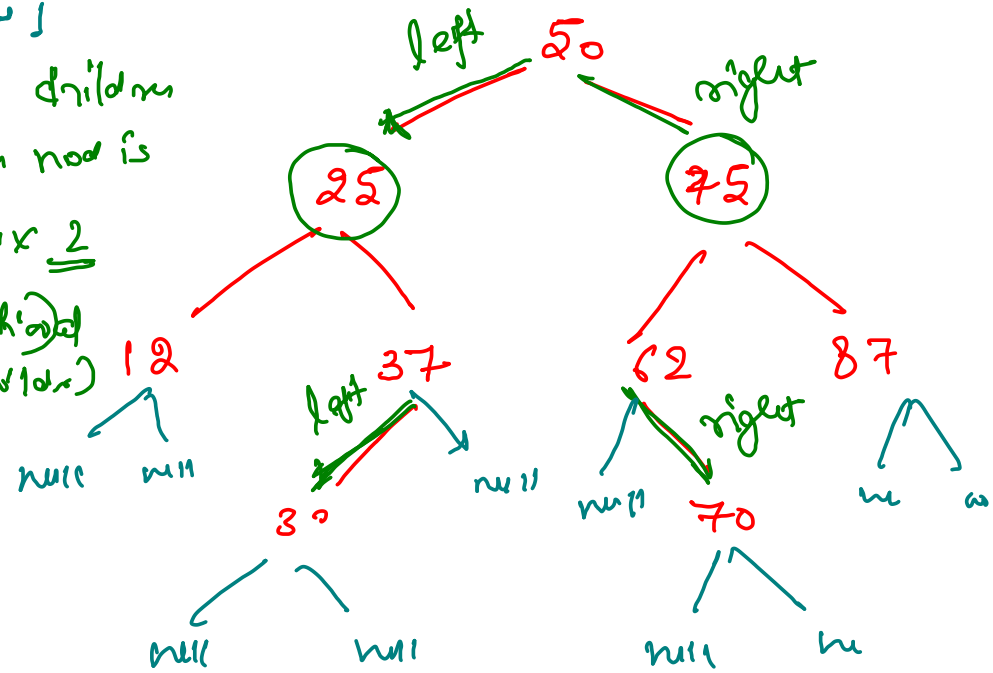
a + (b * c / d)

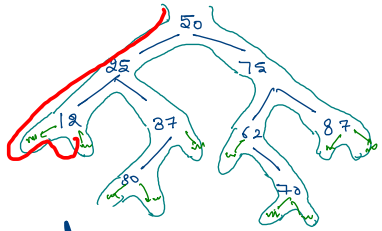



```
Node {
    int data;
    Node left
    Node right
    Constructor
}
```

Binary → 0, 1

No. of children for a node is atmax 2 (Binary children)

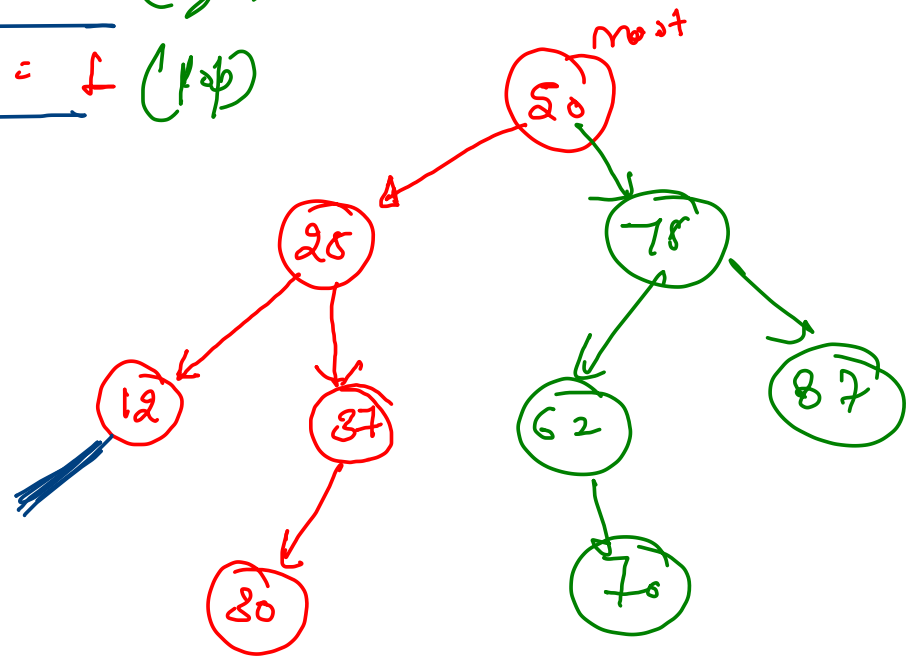




State = -1 (left) → left child, State++, push → -1

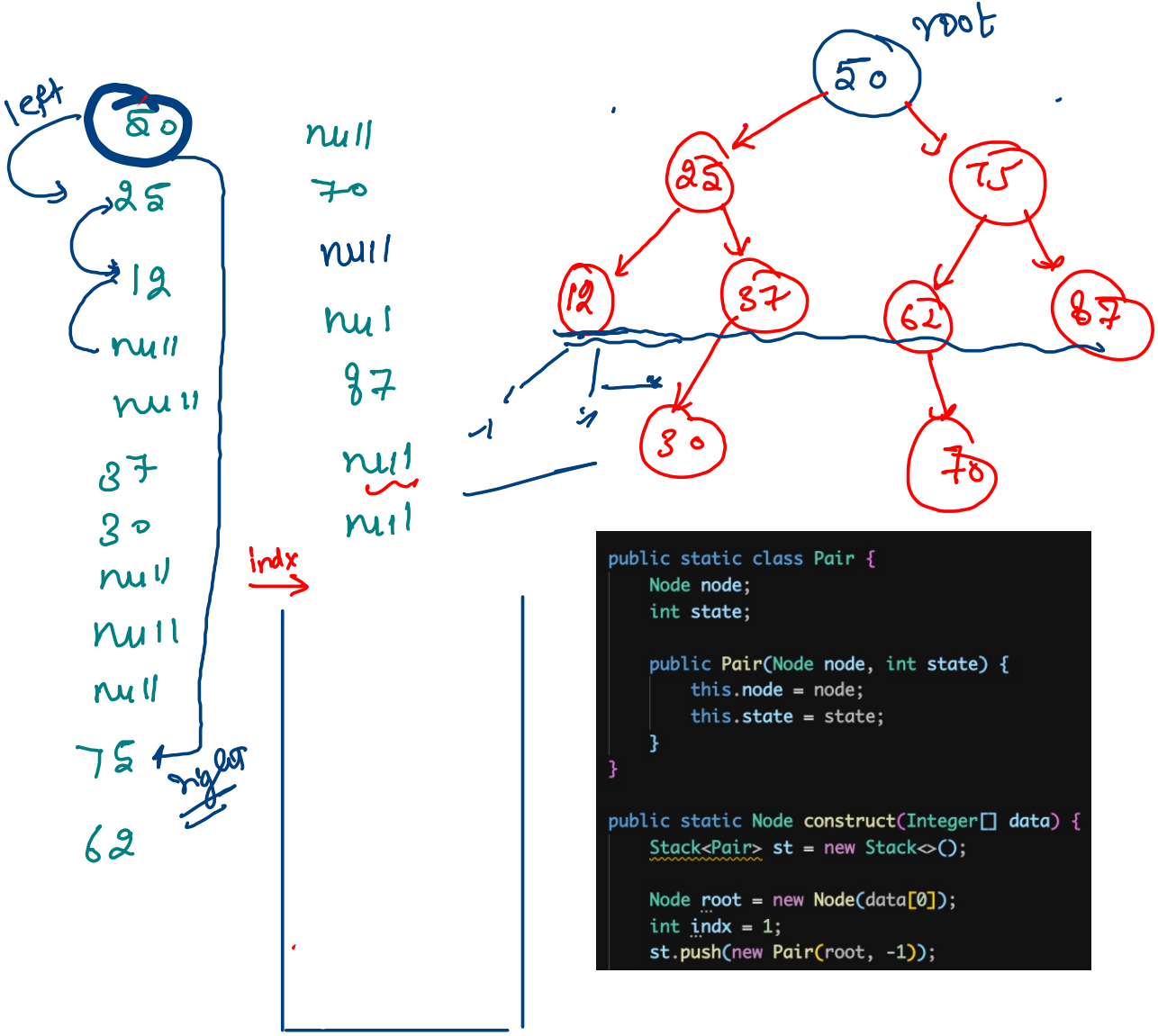
State = 0 (Visited)

State = 1 (Pop)



stack

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




```
public static class Pair {
    Node node;
    int state;

    public Pair(Node node, int state) {
        this.node = node;
        this.state = state;
    }
}

public static Node construct(Integer[] data) {
    Stack<Pair> st = new Stack<>();

    Node root = new Node(data[0]);
    int indx = 1;
    st.push(new Pair(root, -1));
```

```
while(st.size() > 0) {
    Pair p = st.peek();

    if(p.state == -1) {
        // left child addition
        if(data[indx] != null) {
            Node nn = new Node(data[indx]);
            p.node.left = nn;
            st.push(new Pair(nn, -1));
        }
        p.state++;
        indx++;
    } else if(p.state == 0) {
        // right child addition
        if(data[indx] != null) {
            Node nn = new Node(data[indx]);
            p.node.right = nn;
            st.push(new Pair(nn, -1));
        }
        p.state++;
        indx++;
    } else {
        // wipe from out
        st.pop();
    }
}
return root;
```

Display ⟶ (Recursion)

25 ⟵ 50 ⟶ 75

12 ⟵ 25 ⟶ 37

. ⟵ 12 ⟶ .

30 ⟵ 37 ⟶ .

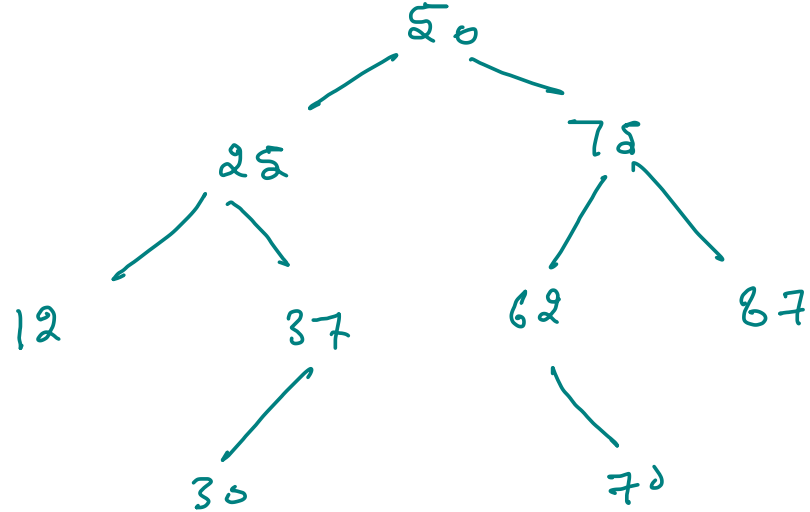
. ⟵ 30 ⟶ .

62 ⟵ 75 ⟶ 87

. ⟵ 62 ⟶ 70

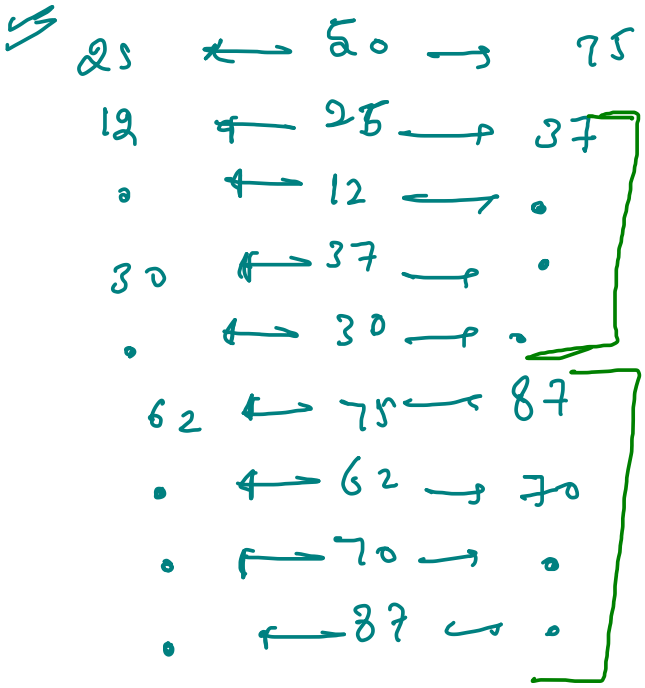
. ⟵ 70 ⟶ .

. ⟵ 87 ⟶ .

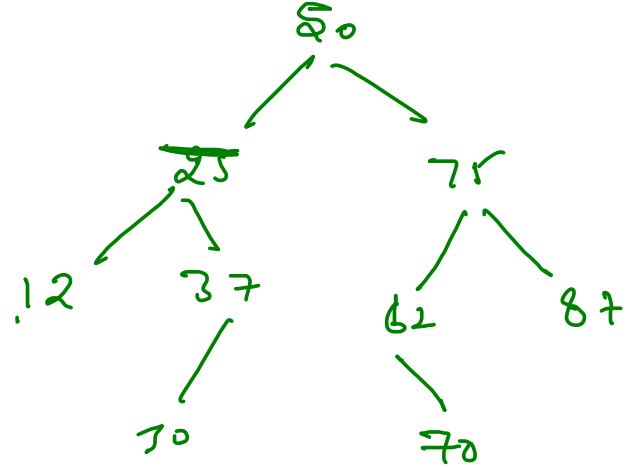


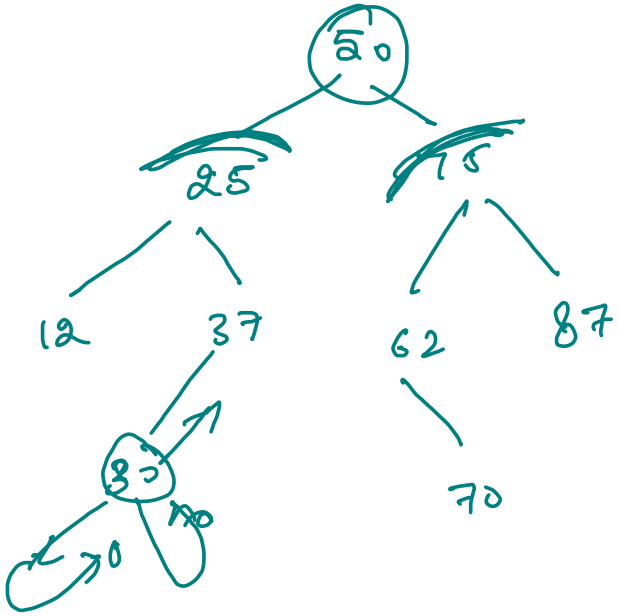
Expectation → display (50) →

faith → display (50.left) → Task 1
display (50.right) → Task 2



Merging → self printing
→ left call
→ right call





Expectation — Size(50) → 9

faith — ls = size(50.left) → 4

rs = size(50.right) → 4

Merging → left call

right call

ls + rs + 1

Base case → if(node == null)

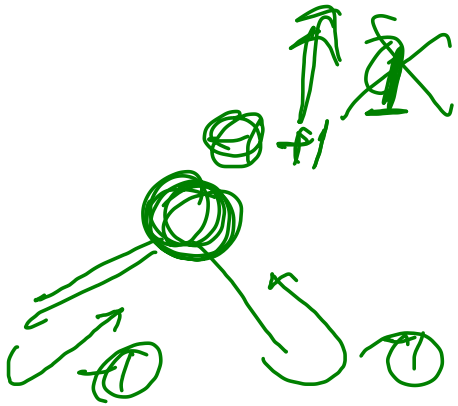
return 0;

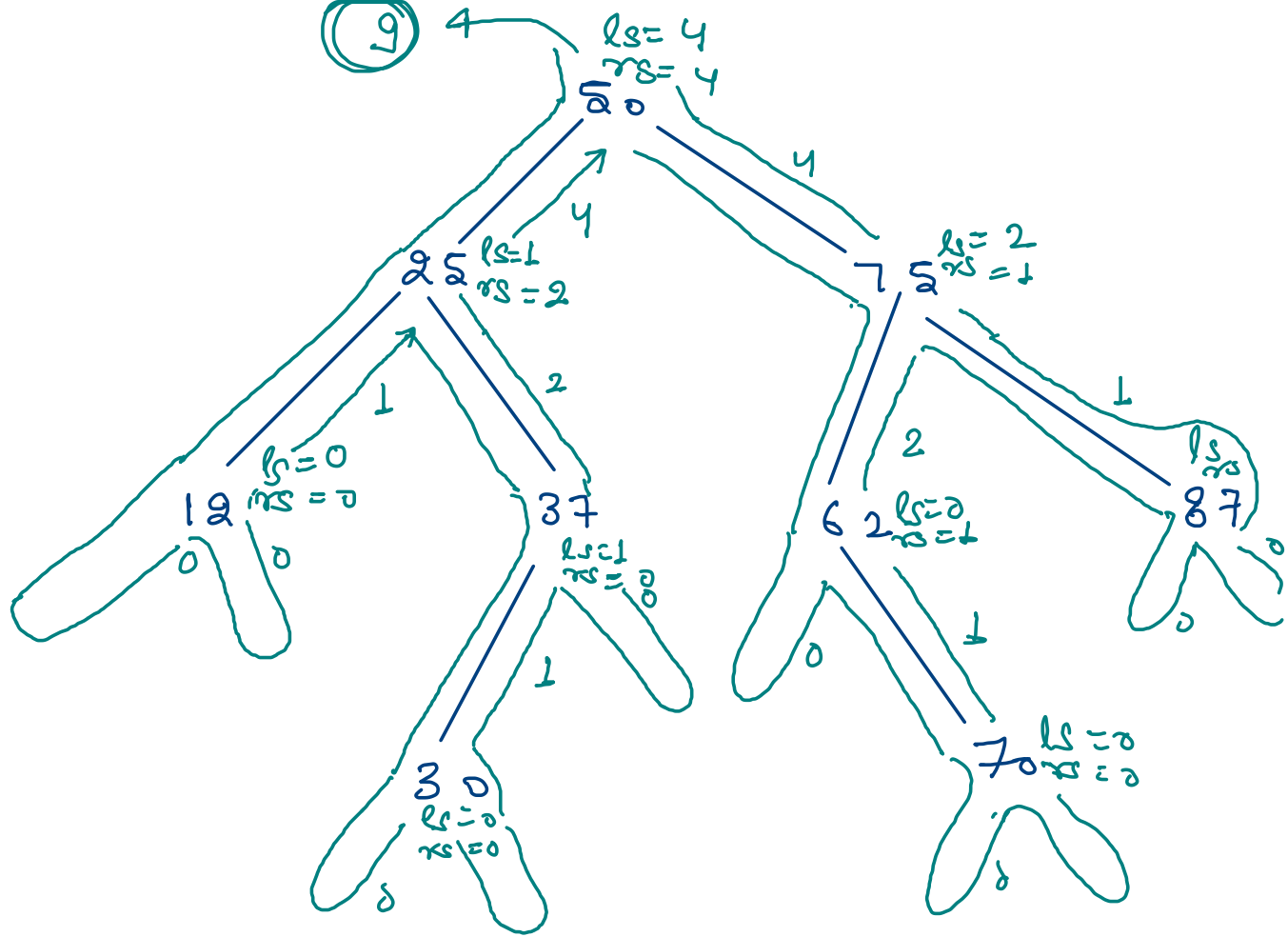
Size.

```java
public static int size(Node node) {
    if(node == null) return 0;

    int lsize = size(node.left);
    int rsize = size(node.right);

    return lsize + rsize + 1;
}
```





Traversal →

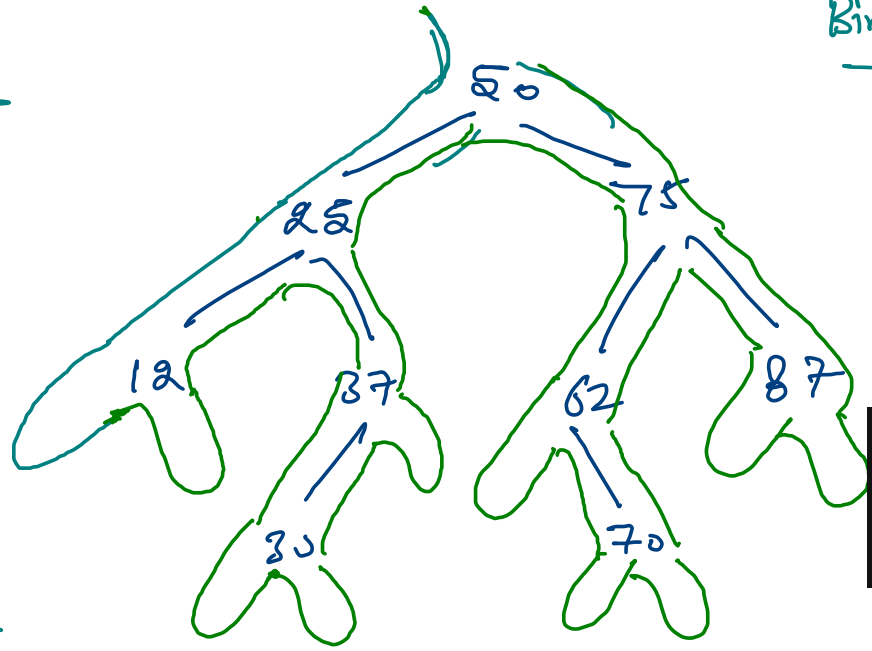
// Pre Area

Call (left)

// In Area

Call (right)

// Post Area



Binary Tree → 1st time → pre

2nd time → In

at snap 3

3rd time → post

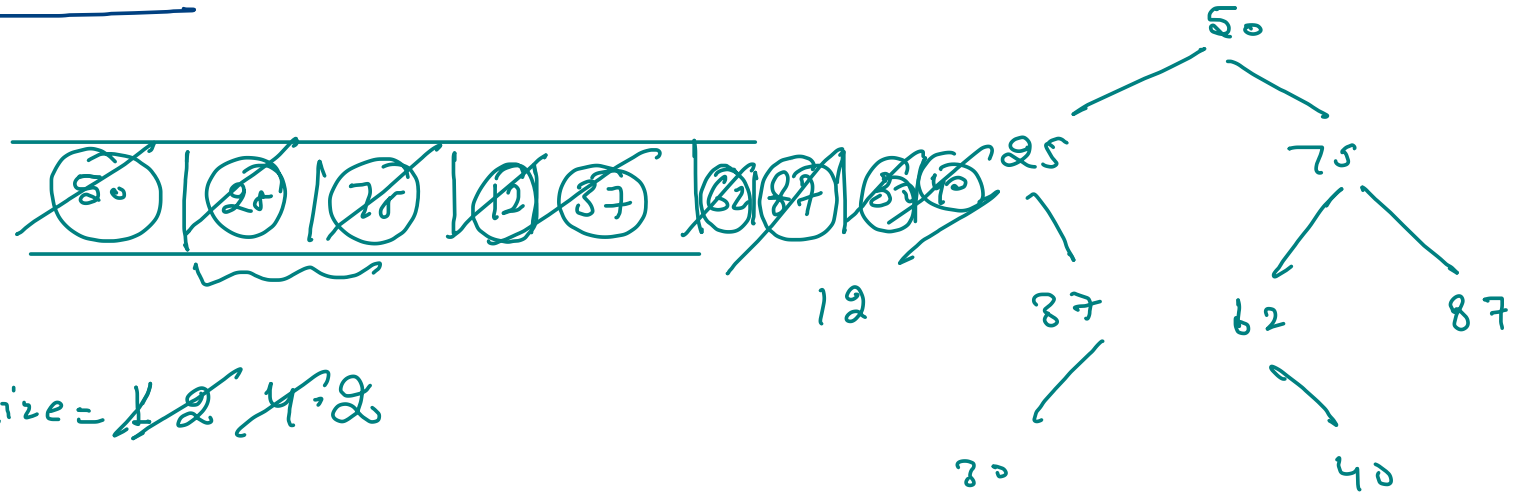
```
Pre Order : 50 25 12 37 30 75 62 70 87
In Order : 12 25 30 37 50 62 70 75 87
Post Order : 12 30 37 25 70 62 87 75 50
```

Pre → 50 25 12 37 30 75 62 70 87

In → 12 25 30 37 50 62 70 75 87

Post → 12 30 37 25 70 62 87 75 50

Level order → line



size = ~~1~~ 2 ~~4~~ 2

50

25 75

12 37 62 87

30 40

get + remove

print

add Chil.