Reverse Level Order Traversal in Binary Trees:

```
class Node {
    constructor(val){
        this.data = val
        this.left = null
        this.right = null
    }

}


const a = new Node("a");
const b = new Node("b");
const c = new Node("c");
const d = new Node("d");
const e = new Node("e");
const f = new Node("f");

a.left = b;
a.right = c;
b.left = d;
b.right = e;
c.right = f;

const ten = new Node(10);
const twenty = new Node(20);
const thirty = new Node(30);
const forty = new Node(40);
const sixty = new Node(60);
const seventy = new Node(70);

ten.left = twenty
ten.right = thirty
twenty.left = forty
twenty.right = sixty
thirty.right = seventy


function reverseLevelOrder(root){
    if (!root) return [];
    return [ ...reverseLevelOrder(root.left), ...reverseLevelOrder(root.right),root.data]
}

console.log(reverseLevelOrder(a))
console.log(reverseLevelOrder(ten))
```

Finding Minimum and Maximum in a Binary Search Tree (BST):

```
class Node {
    constructor(data) {
      this.data = data;
      this.left = null;
      this.right = null;
    }
  }

class BST {
    constructor() {
        this.root = null
    }
    insert(val){
        const newNode = new Node(val)
        if (!this.root) {
            this.root = newNode;
            return;
          }

        let current = this.root
        let prev = null
        while(current){
            if(current.data > val){
                prev = current
                current = current.left
            }
            else if(current.data < val){
                prev = current
                current = current.right
            }
        }
        if(prev.data > val){
           prev.left = newNode
        }
        else if(prev.data < val){
            prev.right = newNode
        }
    }

    min(node = this.root){
        if(!node.left) return node.data
        else return this.min(node.left)

    }
    max(node = this.root){
        if(!node.right) return node.data
        else return this.max(node.right)

    }
}
```

```javascript
const tree = new BST();
tree.insert(5);
tree.insert(3);
tree.insert(7);
tree.insert(2);
tree.insert(4);
tree.insert(6);
tree.insert(8);

console.log(tree.max())
console.log(tree.min())
```