**Tree Includes:**

class Node {

    constructor(val) {

      this.val = 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 treeIncludes = (root, target) => {

    if(!root) return false

    if(root.val === target) return true

    return treeIncludes(root.left, target) || treeIncludes(root.right, target)

}

console.log(treeIncludes(a,"e"))

console.log(treeIncludes(a,"l"))

console.log(treeIncludes(a,"b"))

**Tree Sum:**

class Node {

    constructor(val) {

        this.val = val

        this.left = null

        this.right = null

    }

}

const a = new Node(11);

const b = new Node(2);

const c = new Node(4);

const d = new Node(7);

const e = new Node(21);

const f = new Node(6);

a.left = b;

a.right = c;

b.left = d;

b.right = e;

c.right = f;

const breadthFirstSum = (root) => {

    if(!root) return 0

    let stack = [root]

    let result = 0;

    while(stack.length>0){

        let current = stack.shift()

        result += current.val

        if(current.left) stack.push(current.left)

        if(current.right) stack.push(current.right)

    }

    return result

}

console.log(breadthFirstSum(a));

const recurssiveSum = (root) => {

    if(!root) return 0

    let l = recurssiveSum(root.left)

    let r = recurssiveSum(root.right)

    return root.val + l + r

}

console.log(recurssiveSum(a));

**Reverse Linked List (Iterative and Recursive):**

class Node {

    constructor(data){

        this.data = data

        this.next = null

    }

}

class LinkedList {

    constructor(data){

        this.head = null

    }

    addFirst(data){

      const newNode = new Node(data)

      newNode.next = this.head

      this.head = newNode

    }

    size() {

        let count = 0

        if(!this.head) return count

        let current = this.head

        while(current){

            count ++

            current = current.next

        }

        return count

    }

    print(){

        let current = this.head

        while(current){

            console.log(current.data)

            current = current.next

        }

    }

    reverse() {

        if(!this.head) return null

        let prevPointer = null

        let currentPointer = this.head

        let nextPointer;

        while(currentPointer){

            nextPointer = currentPointer.next;

            currentPointer.next = prevPointer

            prevPointer = currentPointer

            currentPointer = nextPointer

        }

        this.head = prevPointer

    }

    reverseRecurssive(node=this.head, prev=null){

        if(!node){

            this.head = prev

            return

        }

        let nextPointer = node.next

        node.next = prev

        this.reverseRecurssive(nextPointer,node)

    }

}

const linkedlist = new LinkedList();

linkedlist.addFirst(3);

linkedlist.addFirst(13);

linkedlist.addFirst(-8);

linkedlist.addFirst(5);

linkedlist.print()

console.log("================================")

linkedlist.reverse()

linkedlist.print()

console.log("================================")

linkedlist.reverseRecurssive()

linkedlist.print()