// Transaction class to represent a transaction with type, amount, and timestamp

class Transaction {

    constructor(type, amount, timestamp) {

      this.type = type;

      this.amount = amount;

      this.timestamp = timestamp;

    }

  }

  // LinkedListNode class to represent a node in a linked list

  class LinkedListNode {

    constructor(data) {

      this.data = data;

      this.next = null;

    }

  }

  // LinkedList class to represent a linked list for transactions

  class LinkedList {

    constructor() {

      this.head = null;

    }

    // Append a new node with data to the linked list

    append(data) {

      let node = new LinkedListNode(data);

      if (this.head) {

        let current = this.head;

        while (current.next) {

          current = current.next;

        }

        current.next = node;

      } else {

        this.head = node;

      }

    }

    // Display the data of each node in the linked list

    display() {

      let current = this.head;

      while (current) {

        console.log(current.data);

        current = current.next;

      }

    }

  }

  // BankAccount class to represent a bank account with account details and transactions

  class BankAccount {

    constructor(accountNumber, accountHolder, balance = 0) {

      this.accountNumber = accountNumber;

      this.accountHolder = accountHolder;

      this.balance = balance;

      this.transactions = new LinkedList(); // LinkedList to store transactions

    }

  }

  // TreeNode class to represent a node in a binary tree

  class TreeNode {

    constructor(data) {

      this.data = data;

      this.children = [];

    }

  }

  // BinaryTree class to represent a binary tree for accounts

  class BinaryTree {

    constructor() {

      this.root = null;

    }

    // Insert a new node with data into the binary tree

    insert(data) {

      let node = new TreeNode(data);

      this.root ? this.insertRecursive(this.root, node) : (this.root = node);

    }

    // Recursive helper function for inserting nodes into the binary tree

    insertRecursive(node, newNode) {

      node.children.push(newNode);

    }

  }

  // Bank class to represent a bank with accounts and a tree structure

  class Bank {

    constructor() {

      this.accounts = new LinkedList(); // LinkedList to store accounts

      this.accountTree = new TreeNode("Bank"); // TreeNode to represent the overall bank structure

    }

    // Create a new account and add it to the linked list and tree

    createAccount(accountNumber, balance) {

      this.accounts.append({ accountNumber, balance });

      this.addToTree(accountNumber);

    }

    // Add a new account node to the tree

    addToTree(accountNumber) {

      let node = new TreeNode(accountNumber);

      this.accountTree.children.push(node);

    }

    // Transfer money between two accounts

    transferMoney(fromAccount, toAccount, amount) {

      let senderNode = this.findAccountNode(fromAccount, this.accountTree);

      let receiverNode = this.findAccountNode(toAccount, this.accountTree);

      if (senderNode && receiverNode) {

        console.log(`Transferring $${amount} from ${fromAccount} to ${toAccount}`);

      } else {

        console.log("Account not found.");

      }

    }

    // Find the node in the tree corresponding to a given account number

    findAccountNode(accountNumber, node) {

      if (node.data === accountNumber) {

        return node;

      }

      for (let child of node.children) {

        let foundNode = this.findAccountNode(accountNumber, child);

        if (foundNode) {

          return foundNode;

        }

      }

      return null;

    }

    // Check the balance of an account

    checkAmount(accountNumber) {

      console.log(`Checking balance for account ${accountNumber}`);

    }

    // Simulate logging into an account

    login(accountNumber) {

      console.log(`Logging in to account ${accountNumber}`);

    }

  }

  // Sorting Algorithm (Bubble Sort)

  function bubbleSort(arr) {

    let len = arr.length;

    for (let i = 0; i < len - 1; i++) {

      for (let j = 0; j < len - 1; j++) {

        if (arr[j] > arr[j + 1]) {

          let temp = arr[j];

          arr[j] = arr[j + 1];

          arr[j + 1] = temp;

        }

      }

    }

  }

  // Searching Algorithm (Binary Search)

  function binarySearch(sortedArr, target) {

    let low = 0;

    let high = sortedArr.length - 1;

    while (low <= high) {

      let mid = Math.floor((low + high) / 2);

      let guess = sortedArr[mid];

      if (guess === target) {

        return mid;

      } else if (guess < target) {

        low = mid + 1;

      } else {

        high = mid - 1;

      }

    }

    return -1;

  }

  // Example Usage of Sorting and Searching Algorithms

  let numbers = [4, 2, 7, 1, 9, 5];

  console.log("Unsorted Numbers:", numbers);

  bubbleSort(numbers);

  console.log("Sorted Numbers:", numbers);

  let targetNumber = 7;

  let index = binarySearch(numbers, targetNumber);

  console.log(`Index of ${targetNumber} in the sorted array:`, index);

  // Bank operations

  const bank = new Bank();

  bank.createAccount("A1001", 1000);

  bank.createAccount("A1002", 1500);

  console.log("Accounts:");

  bank.accounts.display();

  console.log("\nAccount Tree:");

  console.log(JSON.stringify(bank.accountTree, null, 2));

  bank.transferMoney("A1001", "A1002", 200);

  bank.checkAmount("A1001");

  bank.login("A1002");