Design and Analysis of Algorithms

1 **Write a program non-recursive and recursive program to calculate Fibonacci numbers and analyze their time and space complexity.**

```
public class FibonacciIterative {
   public static long fibonacciIterative(int n) {
      if (n <= 1) {
         return n;
      }

      long a = 0, b = 1;
      for (int i = 2; i <= n; i++) {
         long temp = a;
         a = b;
         b = temp + b;
      }
      return b;
   }

   public static void main(String[] args) {
      int n = 10;
      long result = fibonacciIterative(n);
      System.out.println("Fibonacci(" + n + ") = " +
result);
   }
}
```

```
public class FibonacciRecursive {
   public static long fibonacciRecursive(int n) {
      if (n <= 1) {
         return n;
      }
      return fibonacciRecursive(n - 1) +
fibonacciRecursive(n - 2);
   }

   public static void main(String[] args) {
      int n = 10;
      long result = fibonacciRecursive(n);
      System.out.println("Fibonacci(" + n + ") = " +
result);
   }
}
```

2 **Write a program to implement Huffman Encoding using a greedy strategy.**

```
import java.util.PriorityQueue;
import java.util.HashMap;
import java.util.Map;

class HuffmanNode implements Comparable<HuffmanNode> {
   char data;
   int frequency;
   HuffmanNode left, right;

   public HuffmanNode(char data, int frequency) {
      this.data = data;
      this.frequency = frequency;
      left = right = null;
   }

   @Override
   public int compareTo(HuffmanNode node) {
      return this.frequency - node.frequency;
   }
}

public class HuffmanEncoding {
   public static void printHuffmanCodes(HuffmanNode root, String code) {
      if (root == null) {
         return;
      }
```

```
        if (root.left == null && root.right == null) {
            System.out.println(root.data + ": " + code);
        }

        printHuffmanCodes(root.left, code + "0");
        printHuffmanCodes(root.right, code + "1");
    }

    public static HuffmanNode buildHuffmanTree(Map<Character, Integer> frequencies) {
        PriorityQueue<HuffmanNode> pq = new PriorityQueue<>();

        for (Map.Entry<Character, Integer> entry : frequencies.entrySet()) {
            pq.add(new HuffmanNode(entry.getKey(), entry.getValue()));
        }

        while (pq.size() > 1) {
            HuffmanNode left = pq.poll();
            HuffmanNode right = pq.poll();
            HuffmanNode merged = new HuffmanNode('\0', left.frequency + right.frequency);
            merged.left = left;
            merged.right = right;
            pq.add(merged);
        }

        return pq.poll();
    }

    public static void huffmanCodes(Map<Character, Integer> frequencies) {
        HuffmanNode root = buildHuffmanTree(frequencies);
        printHuffmanCodes(root, "");
    }

    public static void main(String[] args) {
        Map<Character, Integer> frequencies = new HashMap<>();
        frequencies.put('a', 5);
        frequencies.put('b', 9);
        frequencies.put('c', 12);
        frequencies.put('d', 13);
        frequencies.put('e', 16);
        frequencies.put('f', 45);

        huffmanCodes(frequencies);
    }
}
```

3 **Write a program to solve a fractional Knapsack problem using a greedy method.**

```
import java.util.Arrays;
import java.util.Comparator;

class Item {
    int weight;
    int value;
    double valuePerWeight;

    public Item(int weight, int value) {
        this.weight = weight;
        this.value = value;
```

```
        this.valuePerWeight = (double) value / weight;
    }
}
public class FractionalKnapsack {
    public static double getMaxValue(int[] weights, int[] values, int capacity) {
        int n = weights.length;
        Item[] items = new Item[n];

        for (int i = 0; i < n; i++) {
            items[i] = new Item(weights[i], values[i]);
        }
        Arrays.sort(items, Comparator.comparing((Item item) -> item.valuePerWeight).reversed());

        double totalValue = 0.0;
        int remainingCapacity = capacity;

        for (Item item : items) {
            if (remainingCapacity == 0) {
                break;
            }
            int currentWeight = Math.min(item.weight, remainingCapacity);
            totalValue += currentWeight * item.valuePerWeight;
            remainingCapacity -= currentWeight;
        }
        return totalValue;
    }
    public static void main(String[] args) {
        int[] weights = {10, 20, 30};
        int[] values = {60, 100, 120};
        int capacity = 50;

        double maxValue = getMaxValue(weights, values, capacity);
        System.out.println("Maximum value in the knapsack: " + maxValue);
    }
}
```

4 **Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.**

```
public class KnapsackDP {
    public static int knapsack(int capacity, int[] weights, int[] values, int n) {
        int[][] dp = new int[n + 1][capacity + 1];
        for (int i = 0; i <= n; i++) {
            for (int w = 0; w <= capacity; w++) {
                if (i == 0 || w == 0) {
                    dp[i][w] = 0;
                } else if (weights[i - 1] <= w) {
                    dp[i][w] = Math.max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w]);
                } else {
                    dp[i][w] = dp[i - 1][w];
                }
            }
        }
        return dp[n][capacity];
    }
    public static void main(String[] args) {
        int[] values = {60, 100, 120};
        int[] weights = {10, 20, 30};
```

```
        int capacity = 50;
        int n = values.length;
        int maxValue = knapsack(capacity, weights, values, n);
        System.out.println("Maximum value in the knapsack: " + maxValue);
    }
}
```

5 **Design n-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final n-queen‘s matrix.**

```
public class NQueens {
    public static void printBoard(int[][] board) {
        int n = board.length;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print((board[i][j] == 1) ? "Q " : ". ");
            }
            System.out.println();
        }
        System.out.println();
    }

    public static boolean isSafe(int[][] board, int row, int col) {
        int n = board.length;
        for (int i = 0; i < row; i++) {
            if (board[i][col] == 1) {
                return false;
            }
        }
        for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
            if (board[i][j] == 1) {
                return false;
            }
        }
        for (int i = row, j = col; i >= 0 && j < n; i--, j++) {
            if (board[i][j] == 1) {
                return false;
            }
        }
        return true;
    }

    public static boolean solveNQueens(int[][] board, int row) {
        int n = board.length;
        if (row == n) {
            printBoard(board);
            return true;
        }
        boolean hasSolution = false;
        for (int col = 0; col < n; col++) {
            if (isSafe(board, row, col)) {
                board[row][col] = 1;
                hasSolution = solveNQueens(board, row + 1) || hasSolution;
                board[row][col] = 0;
            }
        }
        return hasSolution;
    }
```

```
    public static void main(String[] args) {
        int n = 8;
        int[][] board = new int[n][n];
        board[0][0] = 1;
        if (solveNQueens(board, 1)) {
            System.out.println("Solutions found!");
        } else {
            System.out.println("No solution found.");
        }
    }
}
```

Blockchain Technology

5 **Write a survey report on types of Blockchains and its real time use cases.**

Survey Report: Types of Blockchains and Real-Time Use Cases

Introduction

Blockchain technology has gained significant attention in recent years due to its potential to transform various industries. It offers a secure, transparent, and decentralized way to record and verify transactions. In this survey report, we explore different types of blockchains and their real-time use cases across various sectors.

Types of Blockchains

1. Public Blockchains

Definition: Public blockchains are open to anyone and are not controlled by a single entity. They offer transparency and decentralization.

Real-Time Use Cases:

- Cryptocurrencies: Bitcoin and Ethereum are prime examples of public blockchains used for peer-to-peer transactions and smart contracts.
- Supply Chain Management: Public blockchains provide transparency in supply chain tracking, ensuring the authenticity of products.

2. Private Blockchains

Definition: Private blockchains are restricted to a specific group or organization. Access is controlled, and they are more centralized compared to public blockchains.

Real-Time Use Cases:

- Financial Services: Banks and financial institutions use private blockchains to settle transactions quickly and securely.
- Healthcare: Patient data management and sharing within a network of healthcare providers to maintain data privacy.

3. Consortium Blockchains

Definition: Consortium blockchains are semi-decentralized, controlled by a group of organizations. They offer more trust compared to private blockchains.

Real-Time Use Cases:

- Legal Industry: Law firms and institutions use consortium blockchains for legal document verification and record-keeping.
- Real Estate: Real estate transactions and property management benefit from transparent and secure consortium blockchains.

4. Hybrid Blockchains

Definition: Hybrid blockchains combine elements of both public and private blockchains, offering flexibility in terms of control and privacy.
Real-Time Use Cases:
- Identity Verification: Hybrid blockchains can be used for secure and privacy-preserving identity verification.
- Voting Systems: Ensuring transparency and security in election processes.

Real-Time Use Cases of Blockchains

1. Supply Chain Management
Blockchain Use: Transparency and traceability in the supply chain, reducing fraud and counterfeit products.
Example: Walmart utilizes blockchain to track the source of food products, enhancing food safety.

2. Healthcare
Blockchain Use: Secure and efficient sharing of patient records, ensuring data integrity and privacy.
Example: MedRec is a blockchain system that enables patients to have control over their health data.

3. Financial Services
Blockchain Use: Faster and more secure cross-border payments and smart contracts for financial agreements.
Example: Ripple's XRP ledger facilitates real-time cross-border transactions.

4. Intellectual Property
Blockchain Use: Proving ownership and protecting intellectual property rights.
Example: IPCHAIN Database allows creators to timestamp and protect their creations.

5. Voting Systems
Blockchain Use: Ensuring transparent and secure voting processes, reducing fraud.
Example: Voatz uses blockchain for secure mobile voting.

Conclusion
Blockchain technology offers various types of blockchains to cater to different needs, from public and private to consortium and hybrid blockchains. Real-time use cases of blockchain technology span across multiple industries, providing secure, transparent, and efficient solutions. As the technology continues to evolve, we can expect even more innovative applications of blockchain in the future.

3 **Write a smart contract on a test network, for Bank account of a customer for following operations:**
- **Deposit money**
- **Withdraw Money**
- **Show balance**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Storage {
    address public owner;
    uint256 public balance;

    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Only the owner can perform this operation");
        _;
```

```
    }

    function deposit() external payable {
        require(msg.value > 0, "You must send some ether to deposit.");
        balance += msg.value;
    }

    function withdraw(uint256 amount) external onlyOwner {
        require(amount > 0, "Withdraw amount must be greater than 0");
        require(balance >= amount, "Insufficient balance");

        balance -= amount;
        payable(owner).transfer(amount);
    }

    function getBalance() external view returns (uint256) {
        return balance;
    }
}
```

6	**Write a program to create a Business Network using Hyperledger**

https://www.youtube.com/watch?v=SqgW4YgTjsw