Recursion Question

Count the Number of Digits in a Number
Write a recursive function to count the number of digits in a number.

Print All Permutations of a String
Write a recursive function to generate all permutations of a given string.

Find the Greatest Common Divisor (GCD)
Write a recursive function to find the GCD of two numbers using Euclid’s algorithm.

Tower of Hanoi
Implement the recursive solution for the Tower of Hanoi problem.

Find the Maximum Element in an Array
Write a recursive function to find the maximum element in an array.

Generate All Subsets of a Set
Write a recursive function to generate all subsets of a given set.

N-Queens Problem
Solve the N-Queens problem using recursion.

Permutations of a String with Duplicates
Generate all permutations of a string with duplicate characters.

Solve the Sudoku Puzzle
Implement a recursive function to solve a Sudoku puzzle.

Compute the nth Triangular Number
Write a recursive function to compute the n-th triangular number.

Find All Paths in a Maze
Given a maze, write a recursive function to find all paths from start to end.

Calculate the Binomial Coefficient
Compute the binomial coefficient (n choose k) using recursion.

Generate All Combinations of k Elements from n Elements
Write a recursive function to generate all combinations of k elements from a set of n elements.

Find All Subsets of a Set with a Given Sum
Write a recursive function to find all subsets of a set that sum up to a given value.

exaplain me problem with example in simple way and give java simple and optimized code with time complexity.

Memory updated

Here’s a simple explanation of each problem, along with Java code and time complexity for each one:

1. Count the Number of Digits in a Number

- **Problem**: Find how many digits are in a given number using recursion.
- **Example**: For `n = 12345`, the number of digits is `5`.

```java
public class CountDigits {
    public static int countDigits(int n) {
        if (n == 0) {
            return 0;
        }
        return 1 + countDigits(n / 10);
    }

    public static void main(String[] args) {
        int n = 12345;
        System.out.println(countDigits(n));  // Output: 5
    }
}
```

- **Time Complexity**: $O(\log_{10}(n))$

---

2. Print All Permutations of a String

- **Problem**: Generate all permutations of a given string.
- **Example**: For `s = "ABC"`, the permutations are `ABC, ACB, BAC, BCA, CAB, CBA`.

```java
public class Permutations {
    public static void permute(String str, int l, int r) {
        if (l == r) {
            System.out.println(str);
        } else {
            for (int i = l; i <= r; i++) {
                str = swap(str, l, i);
                permute(str, l + 1, r);
                str = swap(str, l, i);
            }
        }
    }

    public static String swap(String a, int i, int j) {
        char temp;
        char[] charArray = a.toCharArray();
        temp = charArray[i];
        charArray[i] = charArray[j];
        charArray[j] = temp;
        return String.valueOf(charArray);
    }

    public static void main(String[] args) {
        String str = "ABC";
        permute(str, 0, str.length() - 1);
    }
}
```

- **Time Complexity**: O(n * n!)

---

3. Find the Greatest Common Divisor (GCD)

- **Problem**: Find the GCD of two numbers using Euclid's algorithm.
- **Example**: For `a = 56` and `b = 98`, the GCD is `14`.

```java
public class GCD {
    public static int gcd(int a, int b) {
        if (b == 0) {
            return a;
        }
        return gcd(b, a % b);
    }

    public static void main(String[] args) {
        int a = 56, b = 98;
        System.out.println(gcd(a, b));  // Output: 14
    }
}
```

- **Time Complexity**: O(log(min(a, b)))

---

4. Tower of Hanoi

- **Problem**: Move `n` disks from the source to the destination using an auxiliary rod.
- **Example**: For `n = 3`, the sequence of moves is `source → auxiliary, source → destination, auxiliary → destination`.

```java
public class TowerOfHanoi {
    public static void solve(int n, char from, char to, char aux) {
        if (n == 1) {
            System.out.println("Move disk 1 from " + from + " to " + to);
            return;
        }
        solve(n - 1, from, aux, to);
        System.out.println("Move disk " + n + " from " + from + " to " + to);
        solve(n - 1, aux, to, from);
    }

    public static void main(String[] args) {
        int n = 3;
        solve(n, 'A', 'C', 'B');
    }
}
```

- **Time Complexity**: O(2^n)

---

5. Find the Maximum Element in an Array

- **Problem**: Find the maximum element of an array using recursion.
- **Example**: For `arr = {1, 4, 3, -1}`, the maximum element is `4`.

```java
public class MaxElement {
    public static int findMax(int[] arr, int n) {
        if (n == 1) {
            return arr[0];
        }
        return Math.max(arr[n - 1], findMax(arr, n - 1));
    }

    public static void main(String[] args) {
        int[] arr = {1, 4, 3, -1};
        System.out.println(findMax(arr, arr.length));  // Output: 4
    }
}
```

- **Time Complexity**: O(n)

---

6. Generate All Subsets of a Set

- **Problem**: Generate all possible subsets of a given set.
- **Example**: For `set = {1, 2}`, the subsets are `{ {}, {1}, {2}, {1,2} }`.

```java
import java.util.*;

public class Subsets {
    public static void generateSubsets(List<Integer> set, List<Integer> subset, int index)
{
        if (index == set.size()) {
            System.out.println(subset);
            return;
        }
        generateSubsets(set, subset, index + 1);
        subset.add(set.get(index));
        generateSubsets(set, subset, index + 1);
        subset.remove(subset.size() - 1);
    }

    public static void main(String[] args) {
        List<Integer> set = Arrays.asList(1, 2);
        generateSubsets(set, new ArrayList<>(), 0);
    }
}
```

- **Time Complexity**: O(2^n)

Let me know if you need the rest explained or if you want to focus on specific ones!

give remaining

Here are the remaining problems with explanations, Java code, and time complexity:

---

7. N-Queens Problem

- **Problem**: Place `n` queens on an `n x n` chessboard such that no two queens attack each other.
- **Example**: For `n = 4`, one possible solution is:

```css
. Q . .
. . . Q
Q . . .
. . Q .
```

```java
public class NQueens {
    public static boolean isSafe(char[][] board, int row, int col) {
        // Check upper diagonal, lower diagonal, and same row
        for (int i = 0; i < row; i++) {
            if (board[i][col] == 'Q') return false;
            if (col - (row - i) >= 0 && board[i][col - (row - i)] == 'Q') return false;
            if (col + (row - i) < board.length && board[i][col + (row - i)] == 'Q') return
false;
        }
        return true;
    }

    public static void solveNQueens(char[][] board, int row) {
        if (row == board.length) {
            for (char[] r : board) {
                System.out.println(new String(r));
            }
            System.out.println();
            return;
        }
        for (int col = 0; col < board.length; col++) {
            if (isSafe(board, row, col)) {
                board[row][col] = 'Q';
                solveNQueens(board, row + 1);
                board[row][col] = '.';
            }
        }
    }

    public static void main(String[] args) {
        int n = 4;
        char[][] board = new char[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                board[i][j] = '.';
            }
        }
        solveNQueens(board, 0);
    }
}
```

- **Time Complexity**: O(n!)

---

8. Permutations of a String with Duplicates

- **Problem**: Generate all unique permutations of a string with duplicate characters.
- **Example**: For `s = "AAB"`, the permutations are `AAB, ABA, BAA`.

```java
import java.util.*;

public class PermutationsWithDuplicates {
    public static void permuteUnique(char[] chars, int l, Set<String> result) {
        if (l == chars.length) {
            result.add(new String(chars));
        } else {
            for (int i = l; i < chars.length; i++) {
                swap(chars, l, i);
                permuteUnique(chars, l + 1, result);
                swap(chars, l, i); // backtrack
            }
        }
    }

    public static void swap(char[] chars, int i, int j) {
        char temp = chars[i];
        chars[i] = chars[j];
        chars[j] = temp;
    }

    public static void main(String[] args) {
        String s = "AAB";
        Set<String> result = new HashSet<>();
        permuteUnique(s.toCharArray(), 0, result);
        for (String str : result) {
            System.out.println(str);
        }
    }
}
```

- **Time Complexity**: O(n * n!)

---

9. Solve the Sudoku Puzzle

- **Problem**: Solve a `9x9` Sudoku grid where some cells are initially filled.
- **Example**: A partially filled Sudoku grid is solved by filling the empty cells according to Sudoku rules.

```java
public class SudokuSolver {
    public static boolean isValid(char[][] board, int row, int col, char num) {
        for (int i = 0; i < 9; i++) {
            if (board[i][col] == num || board[row][i] == num ||
                board[3 * (row / 3) + i / 3][3 * (col / 3) + i % 3] == num) {
                return false;
            }
        }
        return true;
    }
    public static boolean solveSudoku(char[][] board) {
        for (int row = 0; row < 9; row++) {
            for (int col = 0; col < 9; col++) {
                if (board[row][col] == '.') {
                    for (char num = '1'; num <= '9'; num++) {
                        if (isValid(board, row, col, num)) {
                            board[row][col] = num;
                            if (solveSudoku(board)) return true;
                            board[row][col] = '.'; // backtrack
                        }
                    }
                    return false;
                }
            }
        }
        return true;
    }

    public static void main(String[] args) {
        char[][] board = {
            {'5', '3', '.', '.', '7', '.', '.', '.', '.'},
            {'6', '.', '.', '1', '9', '5', '.', '.', '.'},
            {'.', '9', '8', '.', '.', '.', '.', '6', '.'},
            {'8', '.', '.', '.', '6', '.', '.', '.', '3'},
            {'4', '.', '.', '8', '.', '3', '.', '.', '1'},
            {'7', '.', '.', '.', '2', '.', '.', '.', '6'},
            {'.', '6', '.', '.', '.', '.', '2', '8', '.'},
            {'.', '.', '.', '4', '1', '9', '.', '.', '5'},
            {'.', '.', '.', '.', '8', '.', '.', '7', '9'}
        };
        if (solveSudoku(board)) {
            for (char[] row : board) {
                System.out.println(new String(row));
            }
        } else {
            System.out.println("No solution exists");
        }
    }
}
```

- **Time Complexity**: O(9^(n^2))

---

10. Compute the nth Triangular Number

- **Problem**: Compute the n-th triangular number, which is the sum of all integers from 1 to n.
- **Example**: For `n = 5`, the triangular number is `15` (1 + 2 + 3 + 4 + 5).

```java
public class TriangularNumber {
    public static int triangularNumber(int n) {
        if (n == 1) return 1;
        return n + triangularNumber(n - 1);
    }

    public static void main(String[] args) {
        int n = 5;
        System.out.println(triangularNumber(n));  // Output: 15
    }
}
```

- **Time Complexity**: O(n)

---

11. Find All Paths in a Maze

- **Problem**: Given a maze, find all paths from start to end using recursion.
- **Example**: For a `4x4` maze, find all paths from the top-left to the bottom-right.

```java
public class MazeSolver {
    public static void findPaths(int[][] maze, int x, int y, String path) {
        int n = maze.length;
        if (x == n - 1 && y == n - 1) {
            System.out.println(path);
            return;
        }
        if (x + 1 < n && maze[x + 1][y] == 1) {
            findPaths(maze, x + 1, y, path + "D");
        }
        if (y + 1 < n && maze[x][y + 1] == 1) {
            findPaths(maze, x, y + 1, path + "R");
        }
    }

    public static void main(String[] args) {
        int[][] maze = {
            {1, 0, 0, 0},
            {1, 1, 0, 1},
            {0, 1, 0, 0},
            {1, 1, 1, 1}
        };
        findPaths(maze, 0, 0, "");
    }
}
```

- **Time Complexity**: O(2^n)

---

12. Calculate the Binomial Coefficient

- **Problem**: Compute the binomial coefficient (n choose k).
- **Example**: For `n = 5` and `k = 2`, the binomial coefficient is `10`.

```java
public class BinomialCoefficient {
    public static int binomialCoeff(int n, int k) {
        if (k == 0 || k == n) return 1;
        return binomialCoeff(n - 1, k - 1) + binomialCoeff(n - 1, k);
    }

    public static void main(String[] args) {
        int n = 5, k = 2;
        System.out.println(binomialCoeff(n, k));  // Output: 10
    }
}
```

- **Time Complexity**: O(2^n)

---

13. Generate All Combinations of k Elements from n Elements

- **Problem**: Generate all combinations of `` `k` `` elements from a set of `` `n` `` elements.
- **Example**: For `` `n = 4` `` and `` `k = 2` ``, the combinations are `` `{1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 4}` ``.

java

```java
import java.util.*;

public class Combinations {
    public static void combine(int n, int k, int start, List<Integer> current,
List<List<Integer>> result) {
        if (current.size() == k) {
            result.add(new ArrayList<>(current));
            return;
        }
        for (int i = start; i <= n; i++) {
            current.add(i);
            combine(n, k, i + 1, current, result);
            current.remove(current.size() - 1);
        }
    }

    public static void main(String[] args) {
        int n = 4, k = 2;
        List<List<Integer>> result = new ArrayList<>();
        combine(n, k, 1, new ArrayList<>(), result);
        for (List<Integer> combination : result) {
            System.out.println(combination);
        }
    }
}
```

- **Time Complexity**: O(2^n)

---

14. Find All Subsets of a Set with a Given Sum

- **Problem**: Find all subsets of a set whose elements sum to a given target.
- **Example**: For `` `arr = {2, 3, 5, 7}` ``, and `` `sum = 7` ``, the subsets are `` `{2, 5}` `` and `` `{7}` ``.

java

```java
import java.util.*;

public class SubsetSum {
    public static void findSubsets(int[] arr, int index, List<Integer> current, int sum,
int target) {
        if (sum == target) {
            System.out.println(current);
            return;
        }
        if (index == arr.length || sum > target) return;

        current.add(arr[index]);
        findSubsets(arr, index + 1, current, sum + arr[index], target);
        current.remove(current.size() - 1);
        findSubsets(arr, index + 1, current, sum, target);
    }

    public static void main(String[] args) {
        int[] arr = {2, 3, 5, 7};
        int target = 7;
        findSubsets(arr, 0, new ArrayList<>(), 0, target);
        }
    }
```

- **Time Complexity**: O(2^n)

This completes the set of recursive problems along with explanations and solutions in Java!

ChatGPT can make mistakes. Check important info.