// Program: Compare Recursive and Non-Recursive Fibonacci Implementations

```java
import java.util.Scanner;

public class FibonacciAnalysis {

    // Non-Recursive (Iterative) Method
    static int fibonacciIterative(int n) {
        if (n <= 1)
            return n;

        int prev = 0, curr = 1;
        for (int i = 2; i <= n; i++) {
            int next = prev + curr;
            prev = curr;
            curr = next;
        }
        return curr;
    }

    // Recursive Method
    static int fibonacciRecursive(int n) {
        if (n <= 1)
            return n;
        return fibonacciRecursive(n - 1) + fibonacciRecursive(n - 2);
    }

    // Main Function
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the Fibonacci term (n): ");
        int n = sc.nextInt();

        Runtime runtime = Runtime.getRuntime();

        // ---- ITERATIVE APPROACH ----
        System.out.println("\n--- Iterative Fibonacci ---");

        // Measure memory before
        runtime.gc();
        long beforeUsedMemIter = runtime.totalMemory() - runtime.freeMemory();
        long startTimeIter = System.nanoTime();

        int fibIter = fibonacciIterative(n);

        long endTimeIter = System.nanoTime();
        long afterUsedMemIter = runtime.totalMemory() - runtime.freeMemory();

        long timeTakenIter = endTimeIter - startTimeIter;
        long memoryUsedIter = afterUsedMemIter - beforeUsedMemIter;

        System.out.println("Fibonacci(" + n + ") = " + fibIter);
        System.out.println("Time Taken: " + timeTakenIter + " ns");
        System.out.println("Memory Used: " + memoryUsedIter + " bytes");

        // ---- RECURSIVE APPROACH ----
        System.out.println("\n--- Recursive Fibonacci ---");

        runtime.gc();
        long beforeUsedMemRec = runtime.totalMemory() - runtime.freeMemory();
        long startTimeRec = System.nanoTime();

        int fibRec = fibonacciRecursive(n);
        long endTimeRec = System.nanoTime();
        long afterUsedMemRec = runtime.totalMemory() - runtime.freeMemory();

        long timeTakenRec = endTimeRec - startTimeRec;
        long memoryUsedRec = afterUsedMemRec - beforeUsedMemRec;

        System.out.println("Fibonacci(" + n + ") = " + fibRec);
        System.out.println("Time Taken: " + timeTakenRec + " ns");
        System.out.println("Memory Used: " + memoryUsedRec + " bytes");

        sc.close();
    }
}
```

Output:

```
PS E:\ENGG\7th SEM\DAA\Practical> java FibonacciAnalysis.java
Enter the Fibonacci term (n): 10

--- Iterative Fibonacci ---
Fibonacci(10) = 55
Time Taken: 9500 ns
Memory Used: 0 bytes

--- Recursive Fibonacci ---
Fibonacci(10) = 55
Time Taken: 11600 ns
Memory Used: 0 bytes
```

// Program: Huffman Encoding using Greedy Strategy

```
import java.util.PriorityQueue;
import java.util.Comparator;
import java.util.Scanner;

// A Huffman tree node
class Node {
    char ch;
    int freq;
    Node left, right;

    Node(char ch, int freq) {
        this.ch = ch;
        this.freq = freq;
        this.left = null;
        this.right = null;
    }

    Node(int freq, Node left, Node right) {
        this.ch = '\0'; // Internal node (not a character)
        this.freq = freq;
        this.left = left;
        this.right = right;
    }
}

public class HuffmanEncoding {

    // Recursive function to print Huffman codes from the root
    public static void printCodes(Node root, String code) {
        if (root == null)
            return;

        // Leaf node
        if (root.left == null && root.right == null && Character.isLetter(root.ch)) {
            System.out.println(root.ch + " : " + code);
            return;
        }

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

    // Main function to build Huffman tree and print codes
    public static void buildHuffmanTree(char[] chars, int[] freq) {
        // Step 1: Create a min-heap (priority queue) for Huffman tree nodes
        PriorityQueue<Node> pq = new PriorityQueue<>(Comparator.comparingInt(a ->
a.freq));

        // Step 2: Create a leaf node for each character and add to the queue
        for (int i = 0; i < chars.length; i++) {
            pq.add(new Node(chars[i], freq[i]));
        }

        // Step 3: Build the Huffman Tree
        while (pq.size() > 1) {
            Node left = pq.poll();
            Node right = pq.poll();

            // Combine two smallest nodes
            Node merged = new Node(left.freq + right.freq, left, right);
            pq.add(merged);
        }
        // Step 4: The remaining node is the root node
        Node root = pq.peek();

        System.out.println("\nHuffman Codes:");
        printCodes(root, "");
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter number of characters: ");
        int n = sc.nextInt();

        char[] chars = new char[n];
        int[] freq = new int[n];

        System.out.println("Enter characters and their frequencies:");
        for (int i = 0; i < n; i++) {
            System.out.print("Character " + (i + 1) + ": ");
            chars[i] = sc.next().charAt(0);
            System.out.print("Frequency of " + chars[i] + ": ");
            freq[i] = sc.nextInt();
        }

        buildHuffmanTree(chars, freq);
        sc.close();
    }
}
```

Output:

```
PS E:\ENGG\7th SEM\DAA\Practical> javac HuffmanEncoding.java
PS E:\ENGG\7th SEM\DAA\Practical> java -cp . HuffmanEncoding
Enter number of characters: 4
Enter characters and their frequencies:
Character 1: a
Frequency of a: 2
Character 2: b
Frequency of b: 1
Character 3: c
Frequency of c: 4
Character 4: d
Frequency of d: 3

Huffman Codes:
c : 0
d : 10
b : 110
a : 111
```

```java
// Program: Fractional Knapsack using Greedy Method

import java.util.Arrays;
import java.util.Scanner;

// Class to represent an item
class Item {
    int value, weight;

    Item(int value, int weight) {
        this.value = value;
        this.weight = weight;
    }
}
public class FractionalKnapsack {

    // Function to solve Fractional Knapsack problem
    public static double fractionalKnapsack(int W, Item[] items) {
        // Calculate value/weight ratio and sort items in descending order
        Arrays.sort(items, (a, b) -> Double.compare((double)b.value / b.weight,
(double)a.value / a.weight));

        double totalValue = 0.0;
        int remainingWeight = W;

        for (Item item : items) {
            if (item.weight <= remainingWeight) {
                // Take full item
                totalValue += item.value;
                remainingWeight -= item.weight;
            } else {
                // Take fraction of item
                totalValue += item.value * ((double)remainingWeight / item.weight);
                break; // Knapsack is full
            }
        }
        return totalValue;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter number of items: ");
        int n = sc.nextInt();

        Item[] items = new Item[n];

        System.out.println("Enter value and weight of each item:");
        for (int i = 0; i < n; i++) {
            System.out.print("Item " + (i+1) + " value: ");
            int value = sc.nextInt();
            System.out.print("Item " + (i+1) + " weight: ");
            int weight = sc.nextInt();
            items[i] = new Item(value, weight);
        }
        System.out.print("Enter capacity of the knapsack: ");
        int W = sc.nextInt();

        double maxValue = fractionalKnapsack(W, items);
        System.out.printf("Maximum value in Knapsack = %.2f\n", maxValue);

        sc.close();
    }
}
```

Output:

```
PS E:\ENGG\7th SEM\DAA\Practical> javac FractionalKnapsack.java
PS E:\ENGG\7th SEM\DAA\Practical> java -cp . FractionalKnapsack
Enter number of items: 3
Enter value and weight of each item:
Item 1 value: 100
Item 1 weight: 20
Item 2 value: 30
Item 2 weight: 90
Item 3 value: 60
Item 3 weight: 10
Enter capacity of the knapsack: 50
Maximum value in Knapsack = 166.67
```

// Program: 0-1 Knapsack using Dynamic Programming

```java
import java.util.Scanner;

public class ZeroOneKnapsack {

    // Function to solve 0-1 Knapsack using DP
    public static int knapsack(int[] values, int[] weights, int n, int W) {
        // DP table: rows = items, cols = capacities
        int[][] dp = new int[n + 1][W + 1];

        // Build table dp[][] in bottom-up manner
        for (int i = 0; i <= n; i++) {
            for (int w = 0; w <= W; w++) {
                if (i == 0 || w == 0) {
                    dp[i][w] = 0; // Base case
                } 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 maximum value
        return dp[n][W];
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter number of items: ");
        int n = sc.nextInt();

        int[] values = new int[n];
        int[] weights = new int[n];

        System.out.println("Enter value and weight of each item:");
        for (int i = 0; i < n; i++) {
            System.out.print("Item " + (i + 1) + " value: ");
            values[i] = sc.nextInt();
            System.out.print("Item " + (i + 1) + " weight: ");
            weights[i] = sc.nextInt();
        }

        System.out.print("Enter capacity of the knapsack: ");
        int W = sc.nextInt();

        int maxValue = knapsack(values, weights, n, W);
        System.out.println("Maximum value in 0-1 Knapsack = " + maxValue);

        sc.close();
    }
}
```

Output:

```
PS E:\ENGG\7th SEM\DAA\Practical> javac ZeroOneKnapsack.java
PS E:\ENGG\7th SEM\DAA\Practical> java ZeroOneKnapsack
Enter number of items: 4
Enter value and weight of each item:
Item 1 value: 3
Item 1 weight: 2
Item 2 value: 7
Item 2 weight: 2
Item 3 value: 2
Item 3 weight: 4
Item 4 value: 9
Item 4 weight: 5
Enter capacity of the knapsack: 10
Maximum value in 0-1 Knapsack = 19
```

// Program: Quick Sort Analysis (Deterministic and Randomized)

```java
import java.util.Arrays;
import java.util.Random;
import java.util.Scanner;

public class QuickSortAnalysis {

    // Deterministic Quick Sort (pivot = last element)
    public static void deterministicQuickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = deterministicPartition(arr, low, high);
            deterministicQuickSort(arr, low, pi - 1);
            deterministicQuickSort(arr, pi + 1, high);
        }
    }

    private static int deterministicPartition(int[] arr, int low, int high) {
        int pivot = arr[high]; // last element as pivot
        int i = low - 1;

        for (int j = low; j < high; j++) {
            if (arr[j] <= pivot) {
                i++;
                // Swap arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        // Swap pivot to correct position
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        return i + 1;
    }

    // Randomized Quick Sort (pivot = random element)
    public static void randomizedQuickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = randomizedPartition(arr, low, high);
            randomizedQuickSort(arr, low, pi - 1);
            randomizedQuickSort(arr, pi + 1, high);
        }
    }

    private static int randomizedPartition(int[] arr, int low, int high) {
        Random rand = new Random();
        int pivotIndex = low + rand.nextInt(high - low + 1);
        // Swap random pivot with last element
        int temp = arr[pivotIndex];
        arr[pivotIndex] = arr[high];
        arr[high] = temp;

        return deterministicPartition(arr, low, high);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter size of array: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter array elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        int[] arrDeterministic = Arrays.copyOf(arr, n);
        int[] arrRandomized = Arrays.copyOf(arr, n);

        // Measure deterministic Quick Sort
        long startDet = System.nanoTime();
        deterministicQuickSort(arrDeterministic, 0, n - 1);
        long endDet = System.nanoTime();
        double timeDeterministic = (endDet - startDet) / 1e6; // milliseconds

        // Measure randomized Quick Sort
        long startRand = System.nanoTime();
        randomizedQuickSort(arrRandomized, 0, n - 1);
        long endRand = System.nanoTime();
        double timeRandomized = (endRand - startRand) / 1e6; // milliseconds

        System.out.println("\nSorted array (Deterministic Quick Sort): " +
Arrays.toString(arrDeterministic));
        System.out.println("Time taken (Deterministic): " + timeDeterministic + " ms");

        System.out.println("\nSorted array (Randomized Quick Sort): " +
Arrays.toString(arrRandomized));
        System.out.println("Time taken (Randomized): " + timeRandomized + " ms");

        sc.close();
    }
}
```

Output:

```
PS E:\ENGG\7th SEM\DAA\Practical> javac QuickSortAnalysis.java
PS E:\ENGG\7th SEM\DAA\Practical> java QuickSortAnalysis
Enter size of array: 4
Enter array elements:
3 2 4 1

Sorted array (Deterministic Quick Sort): [1, 2, 3, 4]
Time taken (Deterministic): 0.0065 ms

Sorted array (Randomized Quick Sort): [1, 2, 3, 4]
Time taken (Randomized): 1.8024 ms
```