Assignment 1 Unit 1

CO1 Implement and analyze recursive and nonrecursive Algorithms Analyze

Topics: Implementing and analyzing, fundamental algorithms of Recursive and Nonrecursive problems

1. **Write a program to print the Fibonacci series. (Recursive and Nonrecursive)**

Recursive

```
public class FibonacciRecursive {

    // Recursive method to calculate Fibonacci number
    public static int fibonacci(int n) {
        if (n <= 1) {
            return n; // Base cases: F(0) = 0, F(1) = 1
        }
        return fibonacci(n - 1) + fibonacci(n - 2); // Recursive relation
    }

    public static void main(String[] args) {
        int n = 10; // Number of terms in the Fibonacci series
        System.out.print("Fibonacci Series (Recursive): ");

        // Loop to print the first n Fibonacci numbers
        for (int i = 0; i < n; i++) {
            System.out.print(fibonacci(i) + " ");
        }
    }
}
```

Nonrecursive

```
public class FibonacciNonRecursive

{

// Method to print the Fibonacci series

public static void printFibonacci(int n)

{ if (n <= 0)

{ System.out.println("Please enter a positive number.");

return;

}
```

```
 int first = 0, second = 1;
// First two Fibonacci numbers
System.out.print("Fibonacci Series (Non-Recursive): ");
for (int i = 0; i < n; i++)
{ System.out.print(first + " ");
int next = first + second; // Calculate the next term
first = second;
 // Update first term
 second = next;
// Update second term
} System.out.println();
}
 public static void main(String[] args)
{
 int n = 10;
// Number of terms in the Fibonacci series
printFibonacci(n);
} }
```

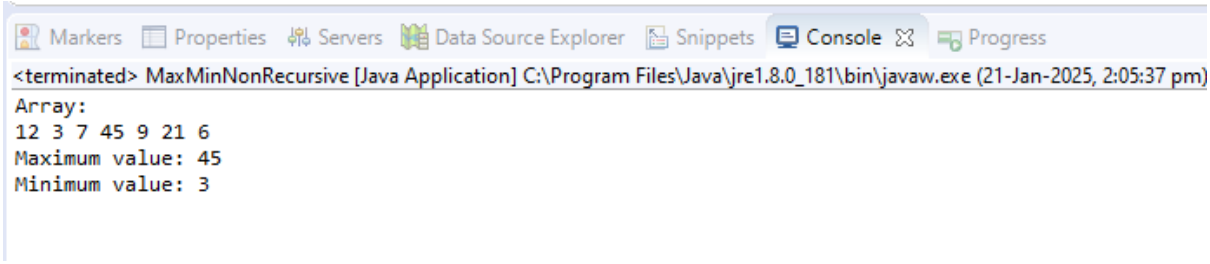
```
<terminated> MergeSort [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\j
Fibonacci Series (Non-Recursive): 0 1 1 2 3 5 8 13 21 34
```

2. **Write a program Maximum and minimum number from an array. (Recursive and Non-recursive)**

Non-recursive

```
public class MaxMinNonRecursive {

   // Method to find the maximum and minimum in an array
   public static void findMaxMin(int[] array) {
      if (array == null || array.length == 0) {
         System.out.println("Array is empty or null!");
         return;
      }

      int max = array[0];
      int min = array[0];

      // Loop through the array to find max and min
```

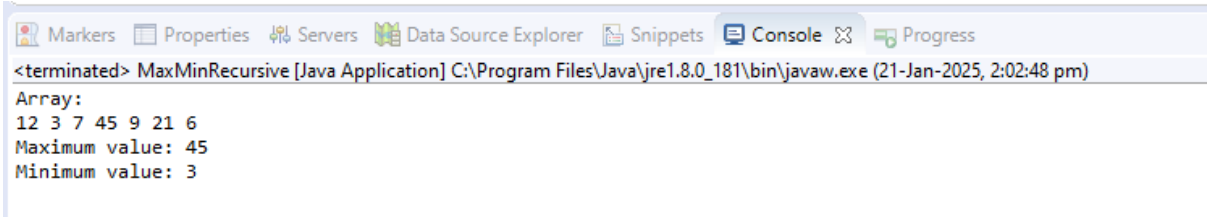
```
        for (int num : array) {
            if (num > max) {
                max = num; // Update max if a larger number is found
            }
            if (num < min) {
                min = num; // Update min if a smaller number is found
            }
        }

        System.out.println("Array: ");
        for (int num : array) {
            System.out.print(num + " ");
        }
        System.out.println("\nMaximum value: " + max);
        System.out.println("Minimum value: " + min);
    }

    public static void main(String[] args) {
        int[] array = {12, 3, 7, 45, 9, 21, 6}; // Example array
        findMaxMin(array);
    }
        }
```



Recursive

```
package sort;

public class MaxMinRecursive {

    // Recursive method to find the maximum value in an array
    public static int findMax(int[] array, int n) {
        if (n == 1) {
            return array[0]; // Base case: If there's only one element
        }
        return Math.max(array[n - 1], findMax(array, n - 1)); // Recursive call
    }

    // Recursive method to find the minimum value in an array
    public static int findMin(int[] array, int n) {
        if (n == 1) {
            return array[0]; // Base case: If there's only one element
        }
        return Math.min(array[n - 1], findMin(array, n - 1)); // Recursive call
```

```
    }

    public static void main(String[] args) {
        int[] array = {12, 3, 7, 45, 9, 21, 6};
        int n = array.length;

        int max = findMax(array, n);
        int min = findMin(array, n);

        System.out.println("Array: ");
        for (int num : array) {
            System.out.print(num + " ");
        }
        System.out.println("\nMaximum value: " + max);
        System.out.println("Minimum value: " + min);
    }
        }
```



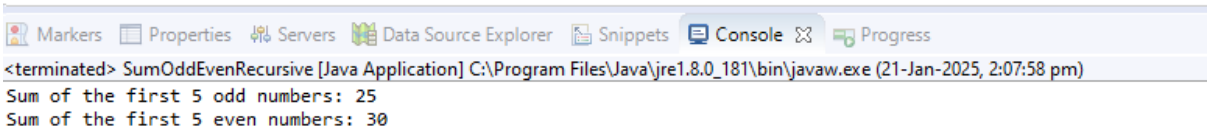
3. **Write a program to find the factorial of a number. (Recursive and Nonrecursive)**

4. **Write a program to find the Sum of the First N Odd & Even Numbers. (Recursive and Non-recursive)**

Recursive

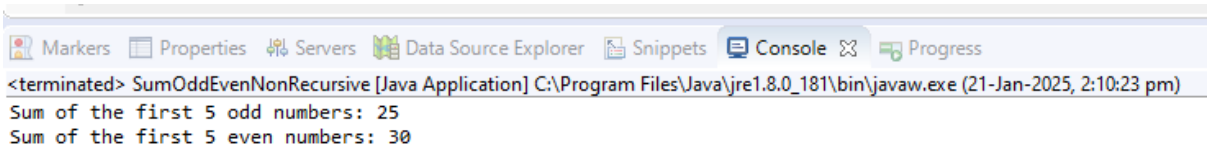
```
public class SumOddEvenRecursive {

    // Recursive method to calculate the sum of the first N odd numbers
    public static int sumOfOddNumbers(int n) {
        if (n == 0) {
            return 0; // Base case: No odd numbers to sum
        }
        return (2 * n - 1) + sumOfOddNumbers(n - 1); // Recursive case
    }

    // Recursive method to calculate the sum of the first N even numbers
    public static int sumOfEvenNumbers(int n) {
        if (n == 0) {
            return 0; // Base case: No even numbers to sum
        }
        return (2 * n) + sumOfEvenNumbers(n - 1); // Recursive case
    }

    public static void main(String[] args) {
```

```
    int n = 5; // Number of terms to calculate

    // Calculate the sums
    int sumOdd = sumOfOddNumbers(n);
    int sumEven = sumOfEvenNumbers(n);

    // Print the results
    System.out.println("Sum of the first " + n + " odd numbers: " + sumOdd);
    System.out.println("Sum of the first " + n + " even numbers: " + sumEven);
}
        }
```



Non-recursive

```
 public class SumOddEvenNonRecursive {

public static void main(String[] args) {
    int n = 5; // Number of terms to calculate

    // Calculate the sum of the first N odd numbers
    int sumOdd = 0;
    for (int i = 1; i <= n; i++) {
        sumOdd += (2 * i - 1); // Formula for the ith odd number
    }

    // Calculate the sum of the first N even numbers
    int sumEven = 0;
    for (int i = 1; i <= n; i++) {
        sumEven += (2 * i); // Formula for the ith even number
    }

    // Print the results
    System.out.println("Sum of the first " + n + " odd numbers: " + sumOdd);
    System.out.println("Sum of the first " + n + " even numbers: " + sumEven);
}
        }
```



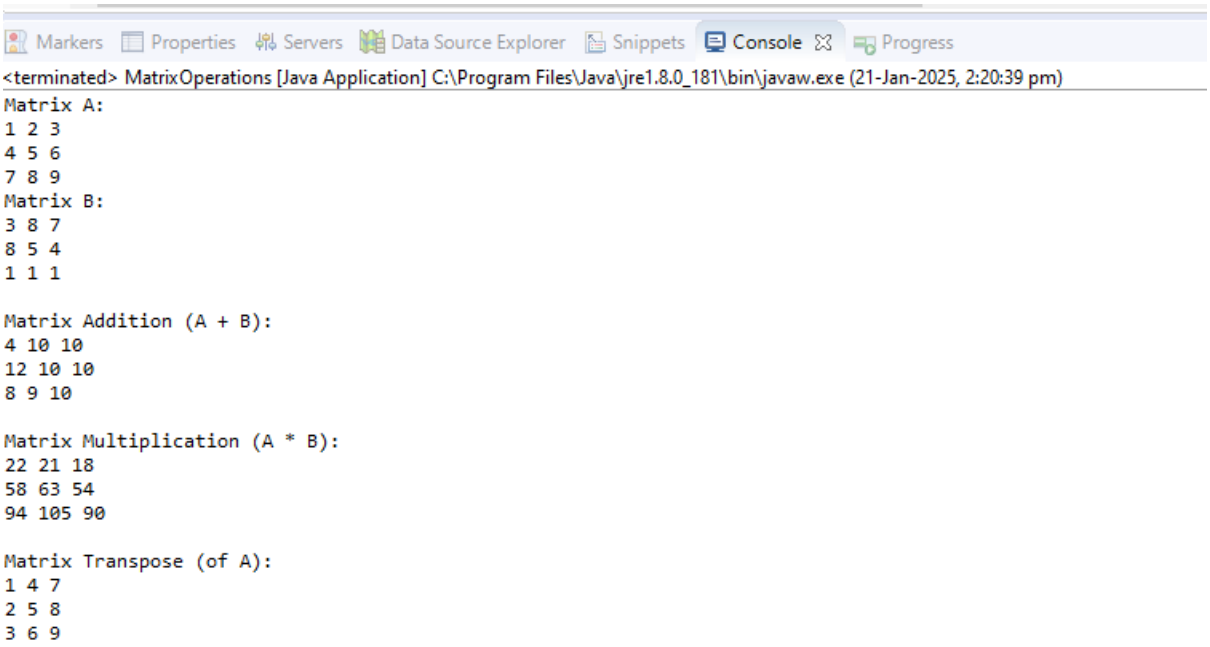
5. **Write a program to add, multiply, and transpose two matrices. (Recursive and Non-recursive)**

Non-recursive

```
public class MatrixOperations {

    // Method for matrix addition
    public static int[][] addMatrices(int[][] A, int[][] B) {
        int rows = A.length;
        int cols = A[0].length;
        int[][] result = new int[rows][cols];

        // Perform addition
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result[i][j] = A[i][j] + B[i][j];
            }
        }
        return result;
    }

    // Method for matrix multiplication
    public static int[][] multiplyMatrices(int[][] A, int[][] B) {
        int rowsA = A.length;
        int colsA = A[0].length;
        int rowsB = B.length;
        int colsB = B[0].length;

        if (colsA != rowsB) {
            throw new IllegalArgumentException("Matrix multiplication is not possible: Number
of columns of A must be equal to number of rows of B.");
        }

        int[][] result = new int[rowsA][colsB];

        // Perform multiplication
        for (int i = 0; i < rowsA; i++) {
            for (int j = 0; j < colsB; j++) {
                result[i][j] = 0;
                for (int k = 0; k < colsA; k++) {
                    result[i][j] += A[i][k] * B[k][j];
                }
            }
        }
```

```
    return result;
}

// Method for matrix transposition
public static int[][] transposeMatrix(int[][] matrix) {
    int rows = matrix.length;
    int cols = matrix[0].length;
    int[][] result = new int[cols][rows];

    // Perform transposition
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            result[j][i] = matrix[i][j];
        }
    }
    return result;
}

// Method to print a matrix
public static void printMatrix(int[][] matrix) {
    for (int[] row : matrix) {
        for (int elem : row) {
            System.out.print(elem + " ");
        }
        System.out.println();
    }
}

public static void main(String[] args) {
    // Example matrices A and B
    int[][] A = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
    };

    int[][] B = {
            {3, 8, 7},
            {8, 5, 4},
            {1, 1, 1}
    };

    // Matrix addition
    System.out.println("Matrix A:");
    printMatrix(A);
    System.out.println("Matrix B:");
    printMatrix(B);

    int[][] additionResult = addMatrices(A, B);
    System.out.println("\nMatrix Addition (A + B):");
```

```
        printMatrix(additionResult);

        // Matrix multiplication
        int[][] multiplicationResult = multiplyMatrices(A, B);
        System.out.println("\nMatrix Multiplication (A * B):");
        printMatrix(multiplicationResult);

        // Matrix transposition
        int[][] transposeResult = transposeMatrix(A);
        System.out.println("\nMatrix Transpose (of A):");
        printMatrix(transposeResult);
    }
}
```



Recursive

```
public class MatrixOperationsRecursive {

    // Recursive method for matrix addition
    public static void addMatrices(int[][] A, int[][] B, int[][] result, int i, int j) {
        if (i >= A.length) {
            return; // Base case: All rows processed
        }
        if (j >= A[0].length) {
            addMatrices(A, B, result, i + 1, 0); // Move to the next row
            return;
        }
        result[i][j] = A[i][j] + B[i][j];
        addMatrices(A, B, result, i, j + 1); // Process next column
    }
```

```
    // Recursive method for matrix multiplication
    public static void multiplyMatrices(int[][] A, int[][] B, int[][] result, int i, int j, int k) {
        if (i >= A.length) {
            return; // Base case: All rows processed
        }
        if (j >= B[0].length) {
            multiplyMatrices(A, B, result, i + 1, 0, 0); // Move to the next row
            return;
        }
        if (k < A[0].length) {
            result[i][j] += A[i][k] * B[k][j];
            multiplyMatrices(A, B, result, i, j, k + 1); // Process next element in row-column
multiplication
        } else {
            multiplyMatrices(A, B, result, i, j + 1, 0); // Move to the next column
        }
    }

    // Recursive method for matrix transposition
    public static void transposeMatrix(int[][] matrix, int[][] result, int i, int j) {
        if (i >= matrix.length) {
            return; // Base case: All rows processed
        }
        if (j >= matrix[0].length) {
            transposeMatrix(matrix, result, i + 1, 0); // Move to the next row
            return;
        }
        result[j][i] = matrix[i][j];
        transposeMatrix(matrix, result, i, j + 1); // Process next column
    }

    // Utility method to print a matrix
    public static void printMatrix(int[][] matrix) {
        for (int[] row : matrix) {
            for (int element : row) {
                System.out.print(element + " ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        int[][] A = {
                {1, 2, 3},
                {4, 5, 6},
                {7, 8, 9}
        };
        int[][] B = {
                {3, 8, 7},
```

```
            {8, 5, 4},
            {1, 1, 1}
        };

        // Matrix addition
        int[][] additionResult = new int[A.length][A[0].length];
        addMatrices(A, B, additionResult, 0, 0);
        System.out.println("Matrix Addition:");
        printMatrix(additionResult);

        // Matrix multiplication
        int[][] multiplicationResult = new int[A.length][B[0].length];
        multiplyMatrices(A, B, multiplicationResult, 0, 0, 0);
        System.out.println("\nMatrix Multiplication:");
        printMatrix(multiplicationResult);

        // Matrix transposition
        int[][] transposeResult = new int[A[0].length][A.length];
        transposeMatrix(A, transposeResult, 0, 0);
        System.out.println("\nMatrix Transpose (of A):");
        printMatrix(transposeResult);
    }
}
```