Matrix Exponentiation

We can find n’th Fibonacci Number in O(Log n) time using Matrix Exponentiation

Same like the binary exponentiation we need to form the initial matrix and same logic
public class MatrixExponentiation {

```
// Function to multiply two matrices
static int[][] multiply(int[][] a, int[][] b) {
    int n = a.length;
    int[][] result = new int[n][n];

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            for (int k = 0; k < n; k++) {
                result[i][j] += a[i][k] * b[k][j];
            }
        }
    }

    return result;
}

// Function to raise a matrix to the power of p using recursion
static int[][] power(int[][] matrix, int p) {
    if (p == 1) {
        return matrix;
    }
    int[][] result;
    if (p % 2 == 0) {
        int[][] halfPower = power(matrix, p / 2);
        result = multiply(halfPower, halfPower);
    } else {
        int[][] halfPower = power(matrix, (p - 1) / 2);
        result = multiply(matrix, multiply(halfPower, halfPower));
    }
    return result;
}
```

```
public static void main(String[] args) {
    int[][] matrix = {{1, 1}, {1, 0}}; // Example matrix
    int exponent = 5; // Example exponent

    int[][] result = power(matrix, exponent);

    // Printing the result
    for (int i = 0; i < result.length; i++) {
        for (int j = 0; j < result[0].length; j++) {
            System.out.print(result[i][j] + " ");
        }
        System.out.println();
    }
}
```

`}

The initial matrix in this example is a 2x2 matrix. It's a common example used in demonstrating matrix exponentiation because it's associated with the Fibonacci sequence.

The matrix in question is:
{{1, 1}, {1, 0}}

| F(n) | | 1 1 | | F(n-1) |
| F(n-1) | = | 1 0 | * | F(n-2) |