Junior Knights OOP Assignment: Recursion

Write the following recursive functions and make calls to each from your main function:

1) Geometric Series

Write a recursive function that determines the sum of a geometric sequence given the value of the first term, the number of terms and the common ratio. The function prototype is below:

```
public static double geoSum(double first, int numterms, double ratio);
```

2) Arithmetic Series

Write a recursive function that determines the sum of an arithmetic sequence given the value of the first term, the number of terms and the common difference. The function prototype is below:

```
public static double arithSum(double first, int numterms, double diff);
```

3) Lucas Numbers

The Lucas, L_n, numbers are defined as follows:

$L_1 = 1$, $L_2 = 3$, $L_n = L_{n-1} + L_{n-2}$, for all $n > 2$.

Write a recursive method that takes in an integer n and returns the n^{th} Lucas number. The method prototype is given below:

```
public static int lucas(int n);
```

4) Tri-Fib Numbers

The Tri-Fib, T_n, numbers are defined as follows:

$T_1 = 1$, $T_2 = 2$, $T_3 = 3$, $T_n = T_{n-1} + T_{n-2} + T_{n-3}$, for all $n > 3$.

Write a recursive method that takes in an integer n and returns the n^{th} Tri-Fib number. The method prototype is given below:

```
public static int triFib(int n);
```

5) Binomial Coefficients

Binomial coefficients, denoted C(n,k), can be computed as follows:

C(n,0) = C(n,n) = 1, for all non-negative integers n.
C(n,k) = C(n-1, k-1) + C(n-1, k), for all integers n > 0 with 0 < k < n.

Write a recursive function that returns the value of C(n, k).

```
// Pre-condition: n ≥ 0, and 0 ≤ k ≤ n
// Post-condition: Returns C(n,k).
int bin_coeff(int n, int k);
```

6) Sum of values in an Array

Write a recursive function that computes the sum of all the values in an array, from index low to index high, inclusive.

```
// Pre-condition: 0 < low, high < array.length
// Post-condition: Returns the sum of array[low..high].
public static int sumArray(int[] array, int low, int high);
```

7) 3n+1

We can create a sequence of integers, given a starting term, n, as follows. If n is even, divide it by 2 to get the next number in the sequence. Otherwise (if n is odd), let 3n+1 be the next term in the sequence. Continue generating terms until 1 is reached. This is the end of the sequence. For example, starting with 6, we generate the sequence 6, 3, 10, 5, 16, 8, 4, 2, 1, a sequence of 9 terms. Write a recursive function with an input value n, so that it returns the length of the sequence starting with n, generated in this fashion.

```
// Pre-condition: n is positive, and produces a sequence
//                with only values less than 1 billion
// Post-condition: Returns the length of the sequence start with
//                 n
public static int threeNPlusOne(int n);
```