Chapter 03, 3.5 – Towers of Hanoi

Chapter 03, 3.1 – Recursive Definitions, Algorithms, and Programs

Chapter 03, 3.2 – Three Questions, Review Problem(s): ***4, 5, 6***

The Towers of Hanoi is an interesting puzzle, with applications in Computer Science.

Try https://www.mathsisfun.com/games/towerofhanoi.html .

How many moves did it take to solve the puzzle for 3 rings? 4 rings?

What is the pattern?

Can you write the ***recursive*** sequence definition for the minimum number of moves, given ***n*** rings? Think MA 116 – Discrete Structures.

$$a_n = a_{n-1} + 1 + a_{n-1}$$

$$a_1 = 1$$

$$a_n = 2 \cdot a_{n-1} + 1, \forall\ int\ n \geq 2$$

This is called a recursive definition because the value of a_n depends on a previous value in the sequence, a_{n-1}.

By the way, the explicit/direct formula would be ***exponential***, since we multiply by 2 each time:

$$a_n = 2^{n-1}, \forall\ int\ n \geq 1$$

How would we write a program to solve the Towers of Hanoi problem? Using recursion. In Java, a recursive method is a method which calls itself.

So, we need to consider how to set up a recursive method, and how we get it to stop calling itself!

Chapter 03 PPT, pg 1-18.

Then, Chapter 03 PPT, pg 37-42.

Run the Towers code.

Stretch goal: Modify the given Towers code so that it prints out the move number.

For example:

```
Input the number of rings: 3
Towers of Hanoi with 3 rings

1 Move ring 1 from peg 1 to peg 3
2 Move ring 2 from peg 1 to peg 2
3 Move ring 1 from peg 3 to peg 2
4 Move ring 3 from peg 1 to peg 3
5 Move ring 1 from peg 2 to peg 1
6 Move ring 2 from peg 2 to peg 3
7 Move ring 1 from peg 1 to peg 3
```

```
public class Towers
{
  public static int numMoves=0;
  public static void doTowers(
         int n,              // Number of rings to move
         int startPeg,       // Peg containing rings to move
         int auxPeg,         // Peg holding rings temporarily
         int endPeg      )   // Peg receiving rings being moved
  {
    if (n == 1) // Base case - Move one ring
    {
      numMoves++;
      System.out.println(numMoves+" Move ring " + n + " from peg " + startPeg
                         + " to peg " + endPeg);
    }
    else
    {
      // Move n - 1 rings from starting peg to auxiliary peg
      doTowers(n - 1, startPeg, endPeg, auxPeg);

      // Move nth ring from starting peg to ending peg
      numMoves++;
      System.out.println(numMoves+" Move ring " + n + " from peg " + startPeg
                         + " to peg " + endPeg);

      // Move n - 1 rings from auxiliary peg to ending peg
      doTowers(n - 1, auxPeg, startPeg, endPeg);
    }
  }
```