

**RECURSION**

Spec

FAQ

Project

Submit

The purpose of this assignment is to give you practice writing programs with recursion.

1. **Trinomial coefficients (brute force).** Write a program `TrinomialBrute.java` that takes two integer command-line arguments  $n$  and  $k$  and computes the corresponding *trinomial coefficient*. The trinomial coefficient  $T(n, k)$  is the coefficient of  $x^{n+k}$  in the expansion of  $(1 + x + x^2)^n$ . For example,

$$(1 + x + x^2)^3 = 1 + 3x + 6x^2 + 7x^3 + 6x^4 + 3x^5 + x^6$$

Thus,  $T(3, 3) = 1$ ,  $T(3, 2) = 3$ ,  $T(3, 1) = 6$ , and  $T(3, 0) = 7$ .

Implement a recursive function `trinomial()` to compute  $T(n, k)$  by using the following recurrence relation:

$$T(n, k) = \begin{cases} 1 & \text{if } n = 0 \text{ and } k = 0 \\ 0 & \text{if } k < -n \text{ or } k > n \\ T(n-1, k-1) + T(n-1, k) + T(n-1, k+1) & \text{otherwise} \end{cases}$$

To do so, organize your program according to the following public API:

```
public class TrinomialBrute {

    // Returns the trinomial coefficient T(n, k).
    public static long trinomial(int n, int k)

    // Takes two integer command-line arguments n and k and prints T(n, k).
    public static void main(String[] args)
}
```

As you will see, this approach is hopeless slow for moderately large values of  $n$  and  $k$ .

```
~/Desktop/recursion> java-introcs TrinomialBrute 3 3
1

~/Desktop/recursion> java-introcs TrinomialBrute 3 2
3

~/Desktop/recursion> java-introcs TrinomialBrute 3 1
6

~/Desktop/recursion> java-introcs TrinomialBrute 3 0
7

~/Desktop/recursion> java-introcs TrinomialBrute 24 12
287134346

~/Desktop/recursion> java-introcs TrinomialBrute 30 0
[takes too long]
```

*Note:* you may assume that  $n$  is a non-negative integer.

*Trinomial coefficients arise in combinatorics. For example,  $T(n, k)$  is the number of permutations of  $n$  symbols, each of which is  $-1$ ,  $0$ , or  $+1$ , which sum to exactly  $k$  and  $T(n, k - n)$  is the number of different ways of randomly drawing  $k$  cards from two identical decks of  $n$  playing cards.*

2. **Trinomial coefficients (dynamic programming).** Write a program `TrinomialDP.java` that takes two integer command-line arguments  $n$  and  $k$  and computes the trinomial coefficient  $T(n, k)$  using *dynamic programming*. To do so, organize your program according to the following public API:

```
public class TrinomialDP {

    // Returns the trinomial coefficient T(n, k).
    public static long trinomial(int n, int k)

    // Takes two integer command-line arguments n and k and prints T(n, k).
    public static void main(String[] args)

}
```

This version should be fast enough to handle larger values of  $n$  and  $k$ .

```
~/Desktop/recursion> java-introcs TrinomialDP 3 0
7

~/Desktop/recursion> java-introcs TrinomialDP 24 12
287134346

~/Desktop/recursion> java-introcs TrinomialDP 30 0
18252025766941

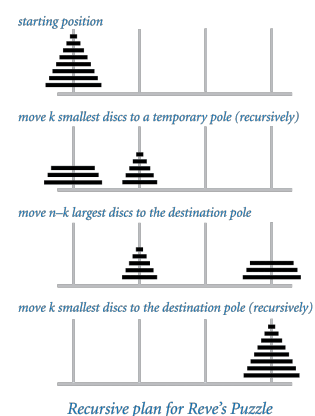
~/Desktop/recursion> java-introcs TrinomialDP 40 0
934837217271732457
```

3. **Reve's puzzle.** *Reve's puzzle* is identical to the *towers of Hanoi* problem, except that there are 4 poles (instead of 3). The task is to move  $n$  discs of different sizes from the starting pole to the destination pole, while obeying the following rules:

- Move only one disc at a time.
- Never place a larger disc on a smaller one.

The following remarkable algorithm, discovered by Frame and Stewart in 1941, transfers  $n$  discs from the starting pole to the destination pole using the fewest moves (although this fact was not proven until 2014).

- Let  $k$  denote the integer nearest to  $n + 1 - \sqrt{2n + 1}$ .
- Transfer (recursively) the  $k$  smallest discs to a single pole other than the start or destination poles.
- Transfer the remaining  $n - k$  disks to the destination pole (without using the pole that now contains the smallest  $k$  discs). To do so, use the algorithm for the 3-pole towers of Hanoi problem.
- Transfer (recursively) the  $k$  smallest discs to the destination pole.



Write a program `RevesPuzzle.java` that takes an integer command-line argument  $n$  and prints a solution to Reve's puzzle. Assume that the discs are labeled in increasing order of size from 1 to  $n$  and that the poles are labeled  $A$ ,  $B$ ,  $C$ , and  $D$ , with  $A$  representing the starting pole and  $D$  representing the destination pole. Here are a few sample executions:

```
~/Desktop/recursion> java-introcs RevesPuzzle 3
Move disc 1 from A to B
Move disc 2 from A to C
Move disc 3 from A to D
Move disc 2 from C to D
Move disc 1 from B to D

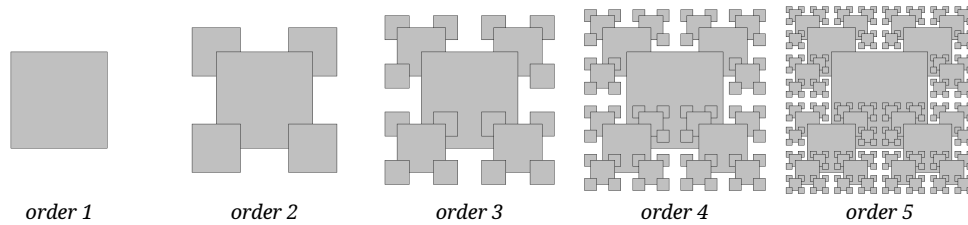
~/Desktop/recursion> java-introcs RevesPuzzle 4
Move disc 1 from A to D
Move disc 2 from A to B
Move disc 1 from D to B
Move disc 3 from A to C
Move disc 4 from A to D
Move disc 3 from C to D
Move disc 1 from B to A
Move disc 2 from B to D
Move disc 1 from A to D

~/Desktop/recursion> java-introcs RevesPuzzle 5
Move disc 1 from A to D
Move disc 2 from A to C
Move disc 3 from A to B
Move disc 2 from C to B
Move disc 1 from D to B
Move disc 4 from A to C
Move disc 5 from A to D
Move disc 4 from C to D
Move disc 1 from B to A
Move disc 2 from B to C
Move disc 3 from B to D
Move disc 2 from C to D
Move disc 1 from A to D
```

*Note:* you may assume that  $n$  is a positive integer.

*Recall that for the towers of Hanoi problem, the minimum number of moves for a 64-disc problem is  $2^{64} - 1$ . With the addition of a fourth pole, the minimum number of moves for a 64-disc problem is reduced to 18,433.*

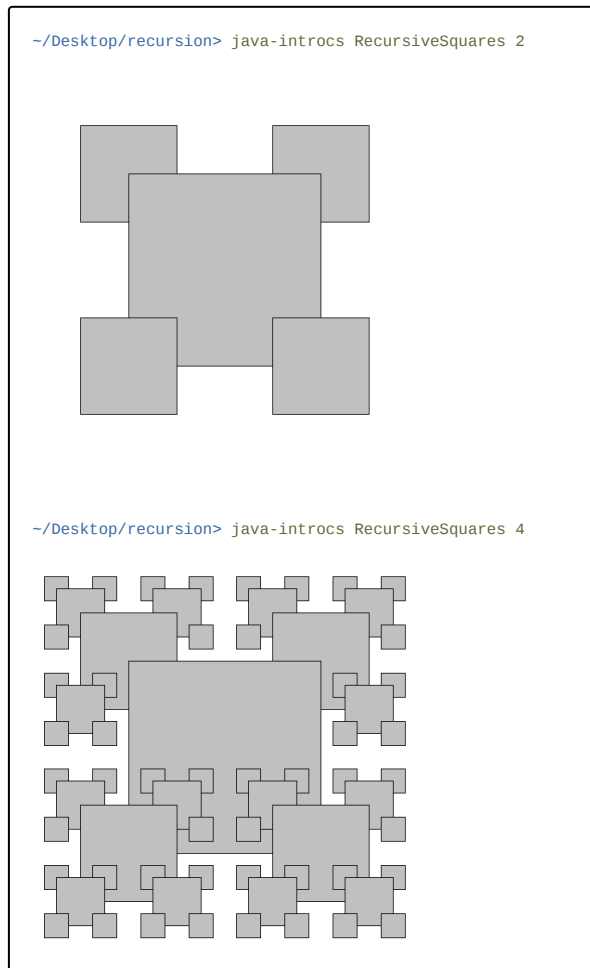
4. **Recursive squares.** Write a program `RecursiveSquares.java` that takes an integer command-line argument  $n$  and plots a recursive square pattern of order  $n$ .



To do so, organize your program according to the following public API:

```
public class RecursiveSquares {  
  
    // Draws a square centered on (x, y) of the given side length  
    // with a light gray background and a black border.  
    public static void drawSquare(double x, double y, double length)  
  
    // Draws a recursive square pattern of order n, centered on (x, y)  
    // of the given side length.  
    public static void draw(int n, double x, double y, double length)  
  
    // Takes an integer command-line argument n and draws a recursive  
    // square pattern of order n, centered on (0.5, 0.5) with side length 0.5.  
    public static void main(String[] args)  
}
```

The largest square is centered on the canvas and has side length 0.5. The side length of each square is one-half that of the next largest square.



*Note:* you may assume that  $n$  is a non-negative integer.

**Submission.** Submit a .zip file containing `TrinomialBrute.java`, `TrinomialDP.java`, `RevesPuzzle.java`, and `RecursiveSquares.java`. You may not call library functions except those in the `java.lang` (such as `Integer.parseInt()` and `Math.sqrt()`). Use only Java features that have already been introduced in the course (e.g., functions and recursion, but not objects).

*This assignment was developed by Kevin Wayne.  
Copyright © 2019.*