Topic 34: Recursion

“Many things of interest to mathematicians or engineers ... have several complicated structures layered on top of one another ... to ... unravel this rich structure ... we view these complex things as a superposition of much simpler ones.

You could imagine trying to work on a screen or tablet with graphical tools putting down broad strokes of color, and then gradually adding finer details. What you are doing is constructing a complex drawing with elementary tools or action.”

Ingrid Daubechies, Ph.D.
President, International Mathematical Union 2011-2014
MacArthur Fellow
Professor, Mathematics and Electrical & Computer Engineering
Inventor of Daubechies wavelets

https://ece.duke.edu/faculty/ingrid-daubechies
What is the output of this program?

public class Cs312 {
    public static void main(String[] args) {
        int n = 6;
        System.out.println(factorial(n));
    }

    public static int factorial(int n) {
        int product = 1;
        for (int i = 1; i <= n; i++) {
            product *= i;
        }
        return product;
    }
}

A. 6
B. 120
C. 720
D. 1080
E. 28750
What is the output of this program?

```java
public class Cs312 {
    public static void main(String[] args) {
        int n = 6;
        System.out.println(factorial(n));
    }

    public static int factorial(int n) {
        if (n <= 1) {
            return 1;
        }
        return n * factorial(n - 1);
    }
}
```
The method calls itself! This is called “recursion.”

```java
public class Cs312 {
    public static void main(String[] args) {
        int n = 6;
        System.out.println(factorial(n));
    }

    public static int factorial(int n) {
        if (n <= 1) {
            return 1;
        }
        return n * factorial(n - 1);
    }
}
```
Recursion

```java
public static int factorial(int n) {
    if (n <= 1) {
        return 1;
    }
    return n * factorial(n - 1);
}
```

factorial(6) = 6 * factorial(5)
= 6 * 5 * factorial(4)
= 6 * 5 * 4 * factorial(3)
= 6 * 5 * 4 * 3 * factorial(2)
= 6 * 5 * 4 * 3 * 2 * factorial(1)
= 6 * 5 * 4 * 3 * 2 * 1
= 720
So…. are these the same?

public static int factorial(int n) {
    int product = 1;
    for (int i = 1; i <= n; i++) {
        product *= i;
    }
    return product;
}

public static int factorial(int n) {
    if (n <= 1) {
        return 1;
    }
    return n * factorial(n - 1);
}
Binary searching for a number, \( n \), in a sorted list

If \( n < \text{midpoint} \):

BinarySearch(\text{left part of list} , n)

If \( n > \text{midpoint} \):

BinarySearch(\text{right part of list} , n)

If \( n == \text{midpoint} \), then return midpoint.
import java.util.Arrays;

public class Cs312 {
    public static void main(String[] args) {
        int[] a = {5, 7, 21, -3, 5, 0, 18, -2, 4, -3};
        Arrays.sort(a);
        int n = 18;
        System.out.println(Arrays.toString(a));
        System.out.println(binarySearch(a, n, 0, a.length - 1));
    }

    // a must be sorted in nondecreasing order
    public static int binarySearch(int[] a, int n, int start, int end) {
        int mid = (start + end) / 2;

        if (start > end) {
            return -1;  // n not found in a
        }
        if (n < a[mid]) {
            return binarySearch(a, n, start, mid - 1);
        }
        if (n > a[mid]) {
            return binarySearch(a, n, mid + 1, end);
        }
        return mid;
    }
}
Fractals & iterated function systems

The Sierpiński Triangle

n = 0  n = 1  n = 2  n = 3  n = 4
Making a Sierpiński Triangle: $n = 0$

$(0, 0)$

$(1, 0)$

$(1/2, \sqrt{3}/2)$
Making a Sierpiński Triangle: $n = 0 \rightarrow n = 1$
Making a Sierpiński Triangle: \( n = 0 \rightarrow n = 1 \)

\[
x_{\text{new}} = a \cdot x + b \cdot y + e \\
y_{\text{new}} = c \cdot x + d \cdot y + f
\]

\[
0 = a \cdot 0 + b \cdot 0 + e \\
0 = c \cdot 0 + d \cdot 0 + f
\]

\[
e = 0 \\
f = 0
\]

\[
1/2 = a \cdot 1 + b \cdot 0 \\
c = 0
\]

\[
1/4 = 1/2 \cdot 1/2 + b \cdot \sqrt{3}/2 \\
\sqrt{3}/4 = d \cdot \sqrt{3}/2
\]

\[
b = 0 \\
d = 1/2
\]

\[
x_{\text{new}} = 1/2 \cdot x \\
y_{\text{new}} = 1/2 \cdot y
\]
Making a Sierpiński Triangle: $n = 0 \rightarrow n = 1$

\[
\begin{align*}
(1/2, \sqrt{3}/2) & \quad (1/2, \sqrt{3}/2) \\
(0, 0) & \quad (1, 0) \\
(1/4, \sqrt{3}/4) & \quad (3/4, \sqrt{3}/4)
\end{align*}
\]

\[
\begin{align*}
x_{\text{new}} &= a \cdot x + b \cdot y + e \\
y_{\text{new}} &= c \cdot x + d \cdot y + f
\end{align*}
\]

\[
\begin{align*}
x_{\text{new}} &= a \cdot 0 + b \cdot 0 + 1/4 \\
y_{\text{new}} &= c \cdot 0 + d \cdot 0 + \sqrt{3}/4
\end{align*}
\]

\[
\begin{align*}
a &= 1/2 \\
b &= 0 \\
c &= 0 \\
d &= 1/2
\end{align*}
\]
Making a Sierpiński Triangle: $n = 0 \rightarrow n = 1$

\[(0, 0) \rightarrow (1/2, \sqrt{3}/2) \rightarrow (3/4, \sqrt{3}/4)\]

\[
x_{\text{new}} = \frac{1}{2} \times x + \frac{1}{2} \\
y_{\text{new}} = \frac{1}{2} \times y
\]
Making a Sierpiński Triangle: $n = 0 \rightarrow n = 1$

So we have 3 transformations:

- $w_1(x, y) = (x/2, y/2)$
- $w_2(x, y) = (x/2 + 1/4, y/2 + \sqrt{3}/4)$
- $w_3(x, y) = (x/2 + 1/2, y/2)$

Applying these 3 transformations together takes us from $n = 0$ to $n = 1$. 
One way to represent a set of filled triangles in Java

If Point is a class that contains two instance variables--double x and double y--then represent this shape as an array of Points:

```
Point[] triangles = {
    new Point(0, 0),
    new Point(1, 0),
    new Point(0.25, Math.sqrt(3)/4),
    new Point(0.5, 0),
    new Point(1, 0),
    new Point(0.75, Math.sqrt(3)/4),
    new Point(0.25, Math.sqrt(3)/4,
    new Point(0.75, Math.sqrt(3)/4,
    new Point(0.5, Math.sqrt(3)/2)};
```

Basically, every group of 3 points together denote the vertices of a particular filled triangle.
public static Point w1(Point point) {
    return new Point(point.x() / 2.0, point.y() / 2.0);
}

public static Point w2(Point point) {
    return new Point(point.x() / 2.0 + 0.25,
                     point.y() / 2.0 + Math.sqrt(3) / 4);
}

public static Point w3(Point point) {
    return new Point(point.x() / 2.0 + 0.5, point.y() / 2.0);
}
public static Point[] transform(Point[] points) {
    Point[] outPoints = new Point[3 * points.length];

    for (int i = 0; i < points.length; i++) {
        outPoints[i] = w1(points[i]);
        outPoints[points.length + i] = w2(points[i]);
        outPoints[2 * points.length + i] = w3(points[i]);
    }

    return outPoints;
}
import java.util.Arrays;

class Point {
    private double x;
    private double y;

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public double x() {
        return x;
    }

    public double y() {
        return y;
    }

    public String toString() {
        return "(" + x + ", " + y + ")";
    }
}
public class Cs312 {
    public static void main(String[] args) {
        Point[] points = {new Point(0, 0), new Point(1, 0),
                          new Point(0.5, Math.sqrt(3) / 2.0)};
        // Point[] newPoints = transformUsingRecursion(points, 9);
        Point[] newPoints = transformUsingForLoop(points, 9);
        System.out.println(Arrays.toString(newPoints));

        DrawingPanel dp = new DrawingPanel(800, 800);
        Graphics g = dp.getGraphics();
        for (int triangleIndex = 0; triangleIndex < newPoints.length / 3;
             triangleIndex++) {
            Point[] triangle = {newPoints[triangleIndex * 3],
                                newPoints[triangleIndex * 3 + 1],
                                newPoints[triangleIndex * 3 + 2]};
            g.fillPolygon(getXs(triangle), getYs(triangle), 3);
        }
    }
}
public static int[] getXs(Point[] points) {
    int[] x = new int[points.length];
    for (int i = 0; i < points.length; i++) {
        x[i] = (int)(points[i].x() * 700 + 50);
    }
    return x;
}

public static int[] getYs(Point[] points) {
    int[] y = new int[points.length];
    for (int i = 0; i < points.length; i++) {
        y[i] = (int)(700 - points[i].y() * 700);
    }
    return y;
}
public static Point w1(Point point) {
    return new Point(point.x() / 2.0, point.y() / 2.0);
}

public static Point w2(Point point) {
    return new Point(point.x() / 2.0 + 0.25,
                     point.y() / 2.0 + Math.sqrt(3) / 4.0);
}

public static Point w3(Point point) {
    return new Point(point.x() / 2.0 + 0.5, point.y() / 2.0);
}

public static Point[] transform(Point[] points) {
    Point[] outPoints = new Point[3 * points.length];
    for (int i = 0; i < points.length; i++) {
        outPoints[i] = w1(points[i]);
        outPoints[points.length + i] = w2(points[i]);
        outPoints[2 * points.length + i] = w3(points[i]);
    }
    return outPoints;
}
public static Point[] transformUsingRecursion(Point[] points, int n) {
    if (n <= 0) {
        return points;
    }
    return transformUsingRecursion(transform(points), n - 1);
}

public static Point[] transformUsingForLoop(Point[] points, int n) {
    Point[] newPoints = points.clone();
    for (int i = 1; i <= n; i++) {
        newPoints = transform(newPoints);
    }
    return newPoints;
}