CMSC 150
INTRODUCTION TO COMPUTING

ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH INTRODUCTION TO PROGRAMMING IN JAVA: AN INTERDISCIPLINARY APPROACH, SEDGEWICK AND WAYNE (PEARSON ADDISON-WESLEY 2007)

LECTURE 7

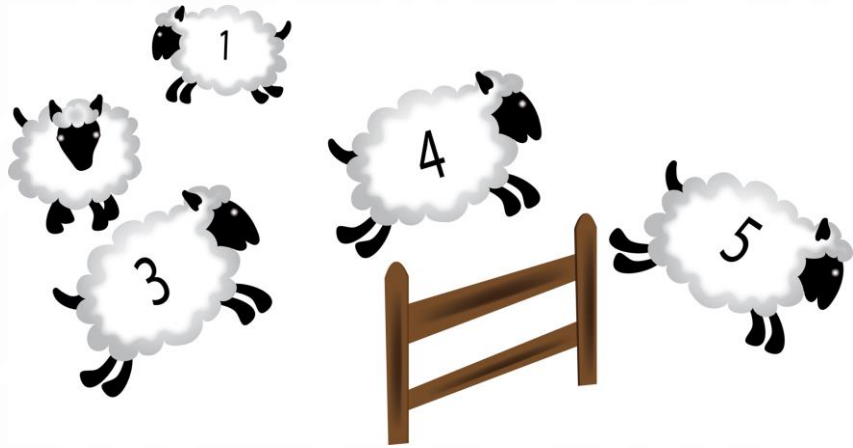
- RECURSION

OVERVIEW

- **Recursion** is an algorithmic technique where a function calls itself directly or indirectly.
- Why learn recursion?
  - New mode of thinking.
  - Powerful programming paradigm.
- Many computations are naturally self-referential.
  - A folder containing files and **other folders**.
  - Mathematical sequences - $f_{n+1} = f_n + 1$ (whole numbers)
  - Exploring mazes – make a step in the maze, then keep **exploring the maze**

EXAMPLE
COUNTING

- Lets take an example of counting from 0
- The next number is the previous number plus 1, or mathematically: $f_n = f_{n-1} + 1$, where $f_0 = 0$
- So lets compute f_5
  - $f_5 = f_4 + 1$, this would be great if we knew f_4, so lets expand it
  - $f_5 = (f_3 + 1) + 1$, then
  - $f_5 = \big((f_2 + 1) + 1\big) + 1$, then
  - $f_5 = \Big(\big((f_1 + 1) + 1\big) + 1\Big) + 1$, then
  - $f_5 = \Big(\big(\big((f_0 + 1) + 1\big) + 1\big) + 1\Big) + 1$, then finally
  - $f_5 = \Big(\big(\big(\big((0) + 1\big) + 1\big) + 1\big) + 1\Big) + 1 = 5$

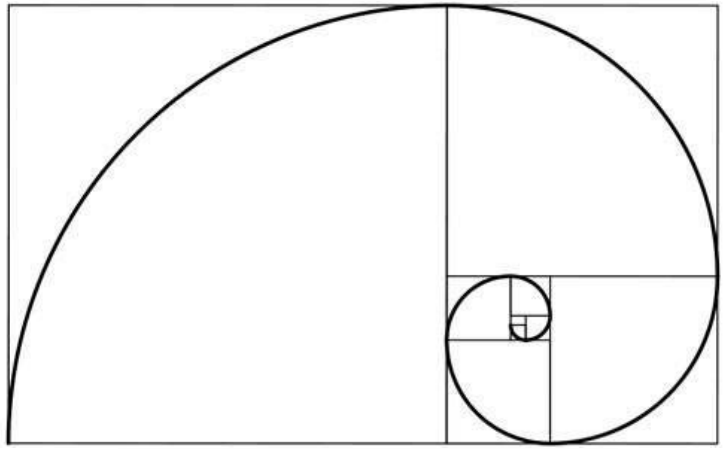


PRACTICE RECURSIVE FORMULAS
FIBONACCI SEQUENCE

- The Fibonacci Sequence is used in various places in mathematics and computer science

  $f_n = f_{n-1} + f_{n-2}$, where $f_0 = 0, f_1 = 1$

- Expand and evaluate f_6, work with a partner and show your work



HOW DO WE DO RECURSION IN CODE?

- Simply call the function within its own body

```
public static void foo() {
    //possibly do some stuff
    foo(); //This example of recursion
    //prossibly do some more stuff
}
```

EXAMPLE
COUNTING

```
//This function counts using recursion
public static int recursiveCount(int n) {
    //f_0 = 0
    if(n == 0)
        return 0;
    //f_n = f_{n-1} + 1
    return recursiveCount(n-1) + 1;
}
```

- Together lets trace
  System.out.println(recursiveCount(3));
- recursiveCount(3)
  - return recursiveCount(2) + 1
    - return recursiveCount(1) + 1;
      - return recursiveCount(0) + 1;
        - return 0;
      - return 0 + 1;
    - return 1 + 1;
  - return 2 + 1;
- 3

PRACTICE RECURSIVE CODE FIBONACCI SEQUENCE

- Write a Java function Fibonacci for

$$f_n = f_{n-1} + f_{n-2}$$

```
public static int Fibonacci(int n) {
    if(n == 0) return 0;
    if(n == 1) return 1;
    return Fibonacci(n-1) + Fibonacci(n-2);
}
```

- Practice tracing Fibonacci(3)
- Fibonacci(3)
  - return Fibonacci(2) + Fibonacci(1);
    - return Fibonacci(1) + Fibonacci(0);
      - return 1;
    - return 1 + Fibonacci(0);
      - return 0;
    - return 1 + 0;
  - return 1 + Fibonacci(1);
    - return 1;
  - return 1 + 1;
- 2

DESIGNING RECURSIVE FUNCTIONS

- Identify the base case
  - The base case is the part of the recursion not defined in terms of itself, e.g., $f_0 = 0, f_1 = 1$
  - This is when the recursion stops! If you forget your base case, then the world will end
    - Really an infinite series of function calls until your computer crashes (if it ever does)
- Identify the recursive process
  - This is the algorithmic process or algorithmic formula
- Write the code

PRACTICE DESIGNING RECURSIVE FUNCTIONS
GREATEST COMMON DENOMINATOR (GCD)

- We saw this in homework
  - For two integers p and q, if q divides p then the GCD of p and q is y
  - Otherwise the GCD of p and q is the same as q and $p\%q$
- Step 1: Identify the base case
  - $q = 0$ implies that the GCD is p
- Step 2: Identify the recursive step
  - $GCD(q, p\%q)$
- Step 3: Code

```
public static int GCD(int p, int q) {
    if(q == 0) return p;
    return GCD(q, p%q);
}
```

RECURSIVE GCD DEMO

```
1.  public class Euclid {
2.    public static int gcd(int p, int q) {
3.      if (q == 0) return p;
4.      else return gcd(q, p % q);
5.    }
6.
7.    public static void main(String[] args) {
8.      int p = Integer.parseInt(args[0]);
9.      int q = Integer.parseInt(args[1]);
10.     System.out.println(gcd(p, q));
11.   }
12. }
```

p = 1272, q = 216

environment

gcd(1272, 216)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
}
```

p = 1272, q = 216

environment

gcd(1272, 216)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
}
```

p = 1272, q = 216

environment

gcd(1272, 216)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
}
```

p = 1272, q = 216

environment

gcd(1272, 216)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
```

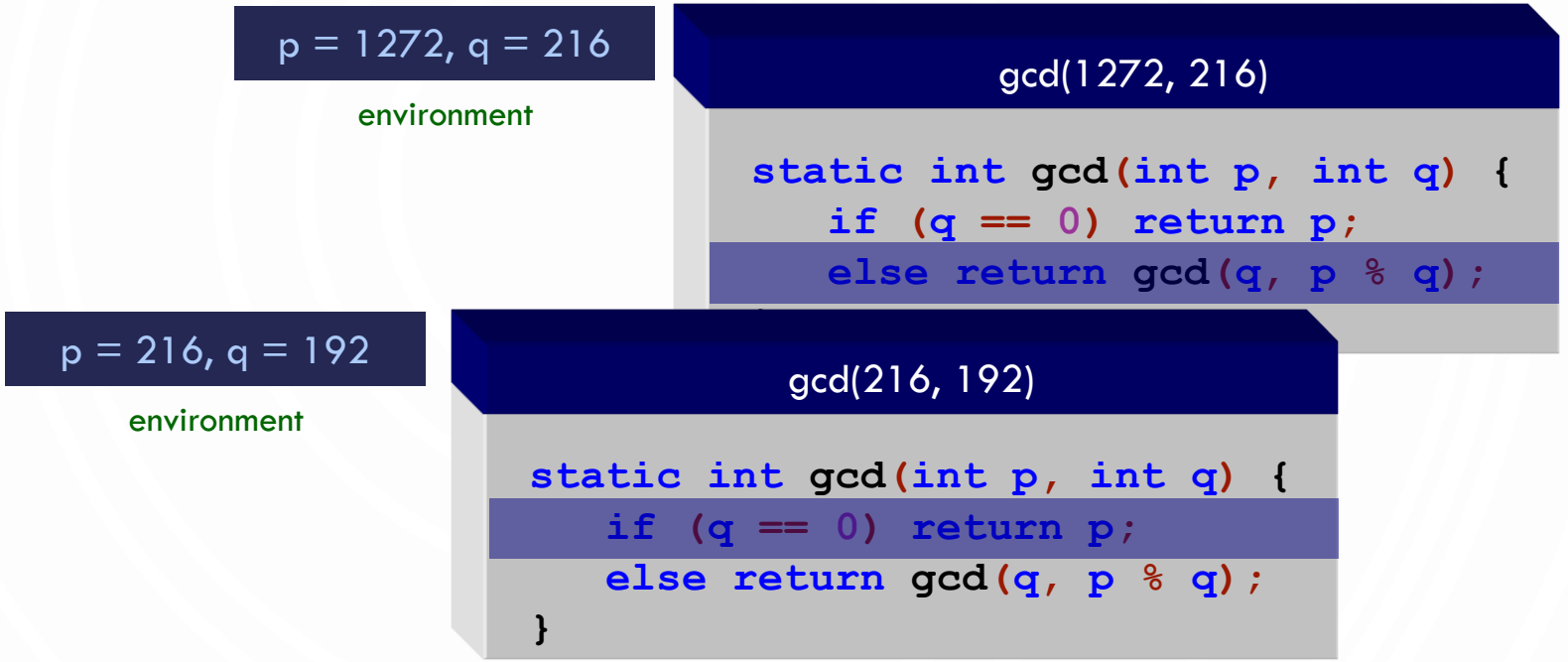
$1272 = 216 \times 5 + 192$

p = 216, q = 192

environment

gcd(216, 192)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
}
```


p = 1272, q = 216
environment
gcd(1272, 216)
static int gcd(int p, int q) {
if (q == 0) return p;
else return gcd(q, p % q);
p = 216, q = 192
environment
gcd(216, 192)
static int gcd(int p, int q) {
if (q == 0) return p;
else return gcd(q, p % q);
}

p = 1272, q = 216

environment

gcd(1272, 216)

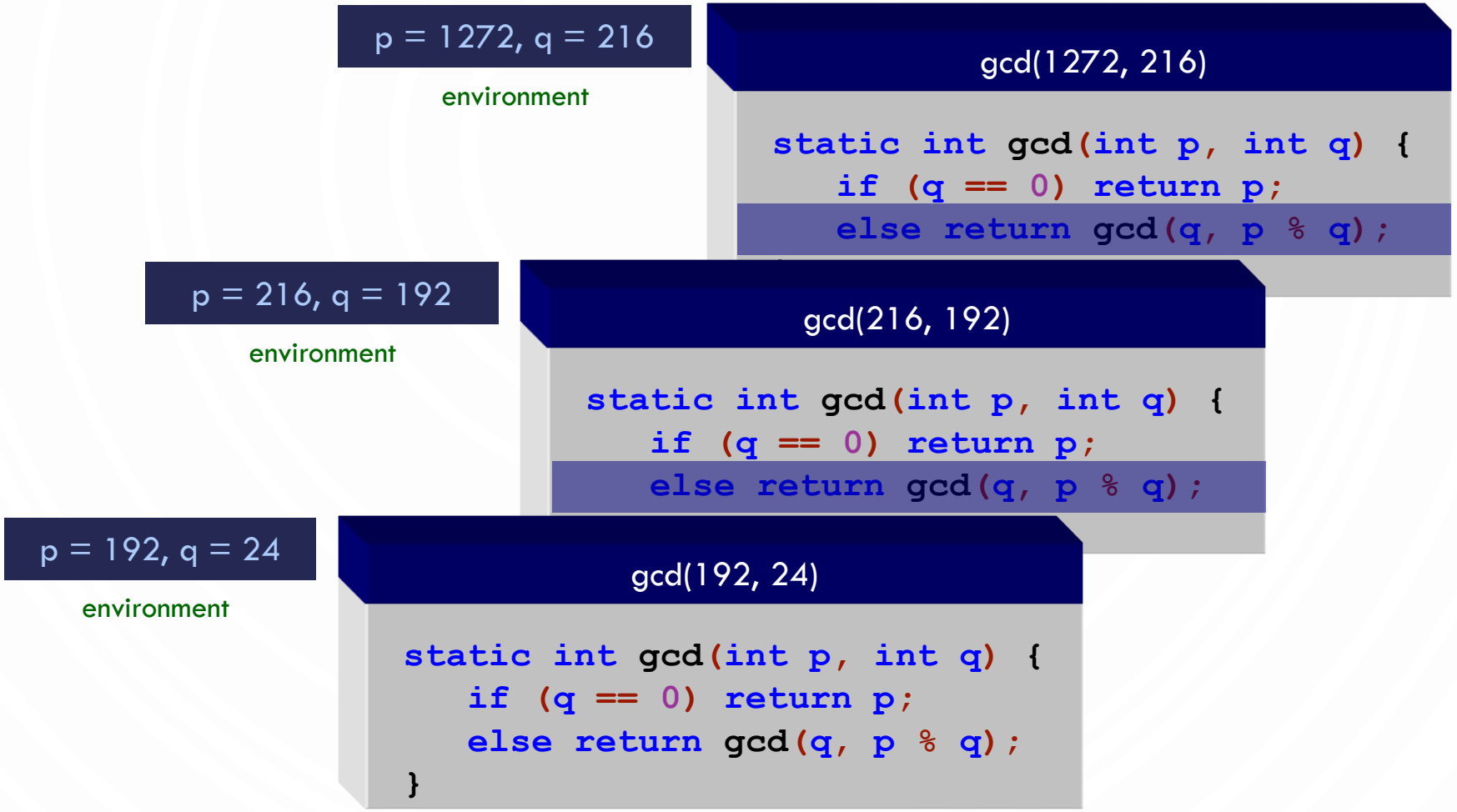
```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
```

p = 216, q = 192

environment

gcd(216, 192)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
}
```


p = 1272, q = 216
environment
gcd(1272, 216)
static int gcd(int p, int q) {
if (q == 0) return p;
else return gcd(q, p % q);
p = 216, q = 192
environment
gcd(216, 192)
static int gcd(int p, int q) {
if (q == 0) return p;
else return gcd(q, p % q);
p = 192, q = 24
environment
gcd(192, 24)
static int gcd(int p, int q) {
if (q == 0) return p;
else return gcd(q, p % q);
}

p = 1272, q = 216

environment

gcd(1272, 216)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
```

p = 216, q = 192

environment

gcd(216, 192)

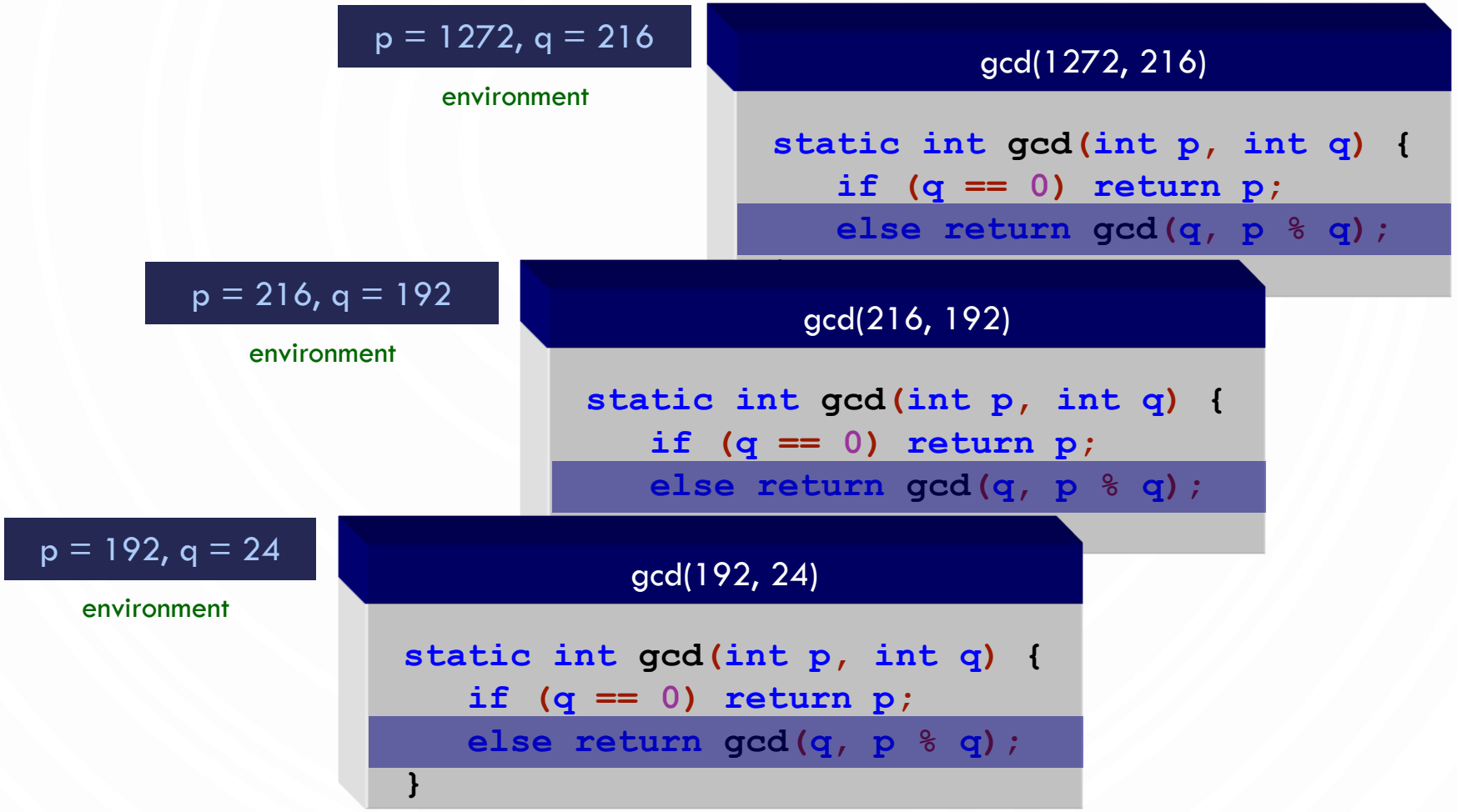
```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
```

p = 192, q = 24

environment

gcd(192, 24)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
}
```


p = 1272, q = 216
environment
gcd(1272, 216)
static int gcd(int p, int q) {
if (q == 0) return p;
else return gcd(q, p % q);
p = 216, q = 192
environment
gcd(216, 192)
static int gcd(int p, int q) {
if (q == 0) return p;
else return gcd(q, p % q);
p = 192, q = 24
environment
gcd(192, 24)
static int gcd(int p, int q) {
if (q == 0) return p;
else return gcd(q, p % q);
}

p = 1272, q = 216

environment

gcd(1272, 216)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
```

p = 216, q = 192

environment

gcd(216, 192)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
```

p = 192, q = 24

environment

gcd(192, 24)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
```

p = 24, q = 0

environment

gcd(24, 0)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
}
```

p = 1272, q = 216

environment

gcd(1272, 216)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
```

p = 216, q = 192

environment

gcd(216, 192)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
```

p = 192, q = 24

environment

gcd(192, 24)

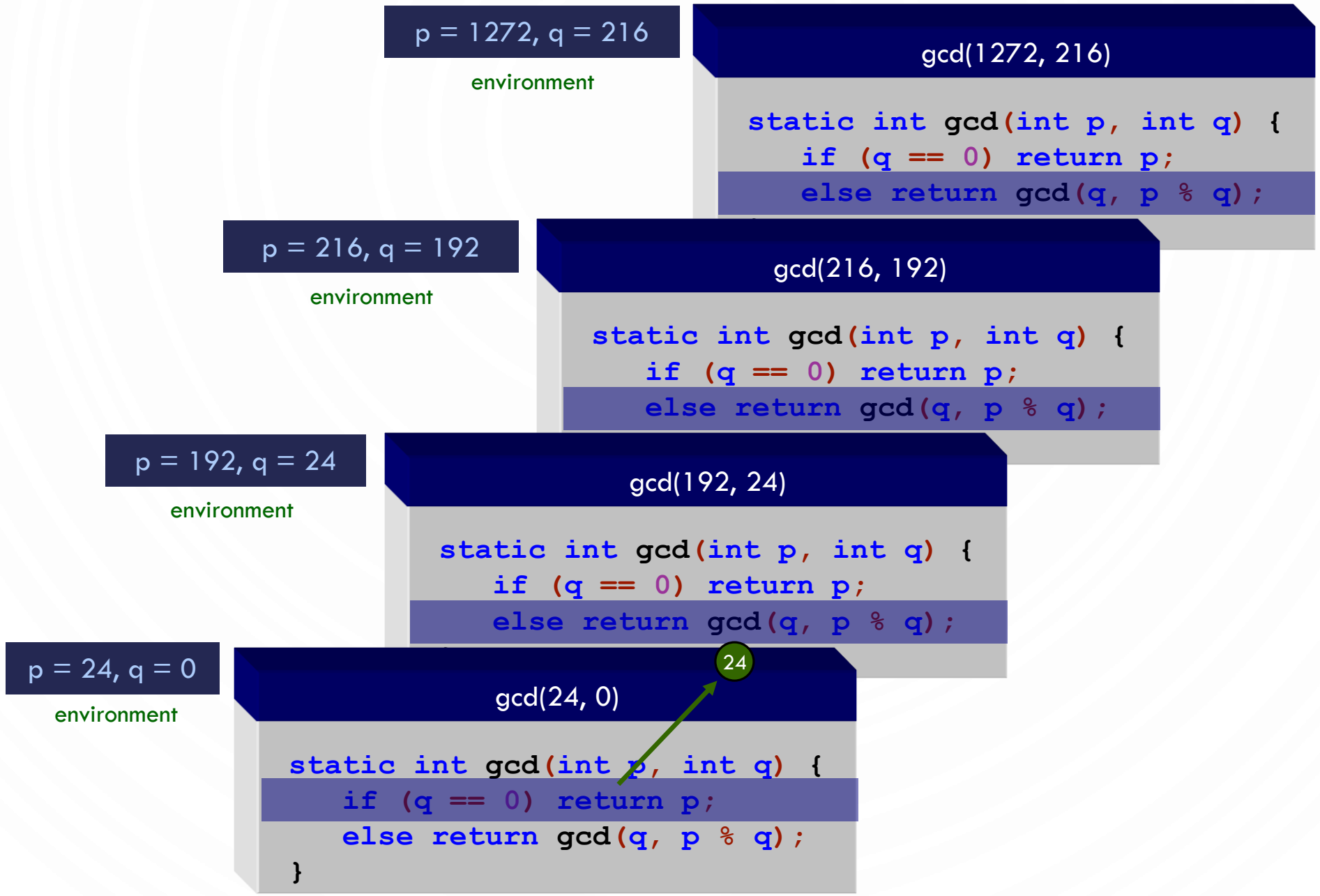
```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
```

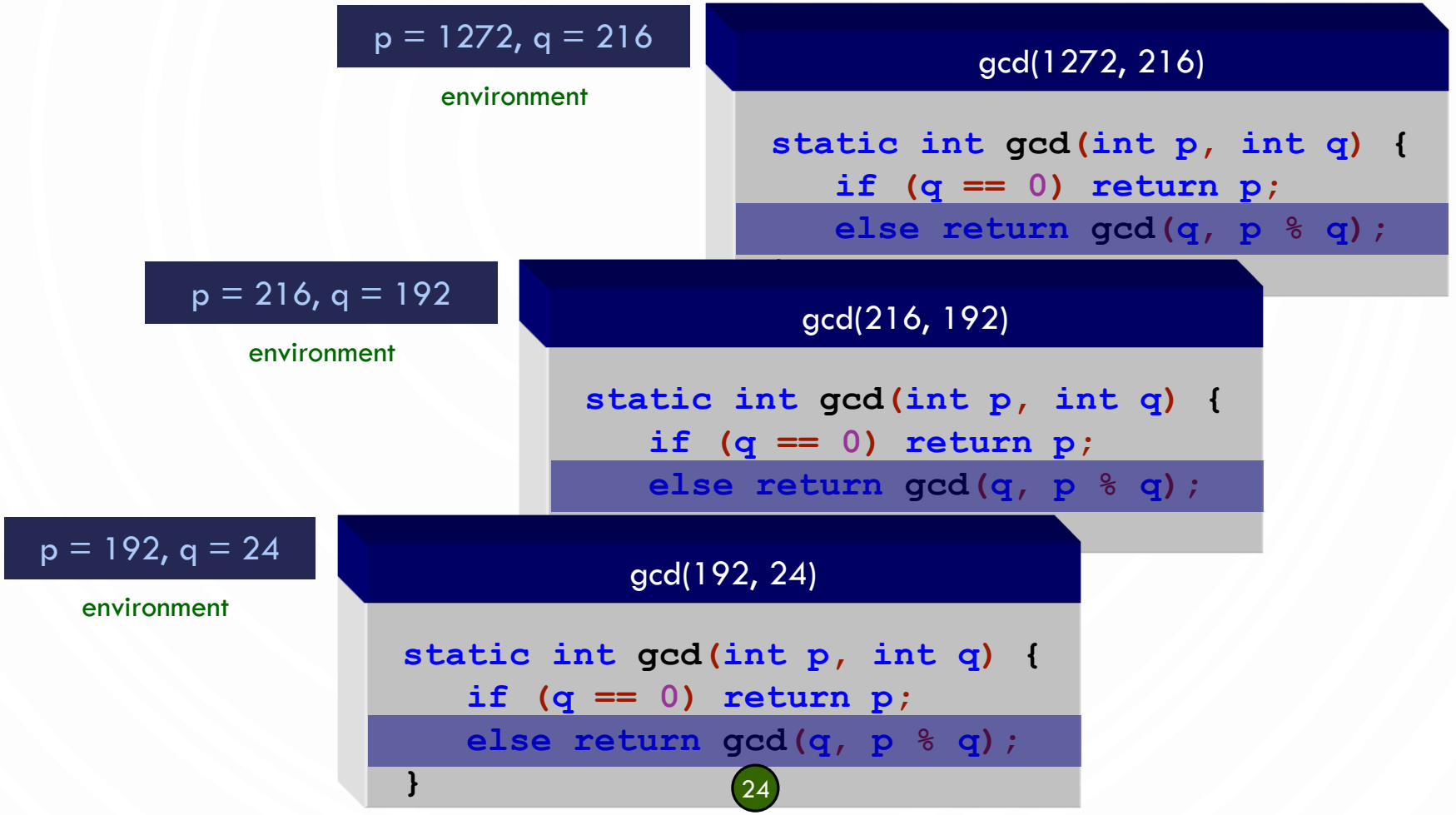
p = 24, q = 0

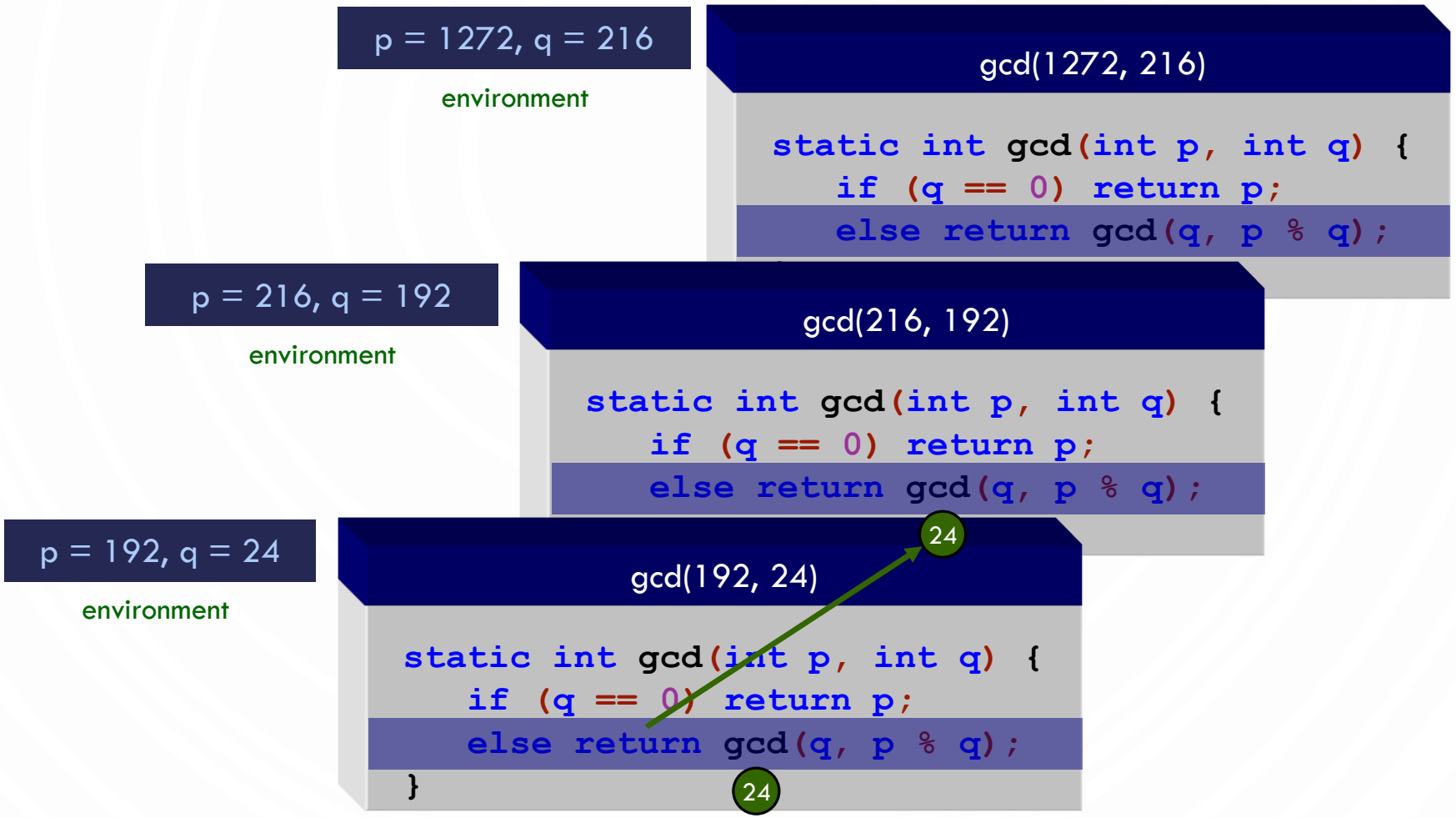
environment

gcd(24, 0)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
}
```


p = 1272, q = 216
environment
gcd(1272, 216)
static int gcd(int p, int q) {
if (q == 0) return p;
else return gcd(q, p % q);
p = 216, q = 192
environment
gcd(216, 192)
static int gcd(int p, int q) {
if (q == 0) return p;
else return gcd(q, p % q);
p = 192, q = 24
environment
gcd(192, 24)
static int gcd(int p, int q) {
if (q == 0) return p;
else return gcd(q, p % q);
p = 24, q = 0
environment
gcd(24, 0)
24
static int gcd(int p, int q) {
if (q == 0) return p;
else return gcd(q, p % q);
}


p = 1272, q = 216
environment
gcd(1272, 216)
static int gcd(int p, int q) {
if (q == 0) return p;
else return gcd(q, p % q);
p = 216, q = 192
environment
gcd(216, 192)
static int gcd(int p, int q) {
if (q == 0) return p;
else return gcd(q, p % q);
p = 192, q = 24
environment
gcd(192, 24)
static int gcd(int p, int q) {
if (q == 0) return p;
else return gcd(q, p % q);
}
24


p = 1272, q = 216
environment
gcd(1272, 216)
static int gcd(int p, int q) {
if (q == 0) return p;
else return gcd(q, p % q);
p = 216, q = 192
environment
gcd(216, 192)
static int gcd(int p, int q) {
if (q == 0) return p;
else return gcd(q, p % q);
p = 192, q = 24
environment
gcd(192, 24)
static int gcd(int p, int q) {
if (q == 0) return p;
else return gcd(q, p % q);
}
24
24

p = 1272, q = 216

environment

gcd(1272, 216)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
```

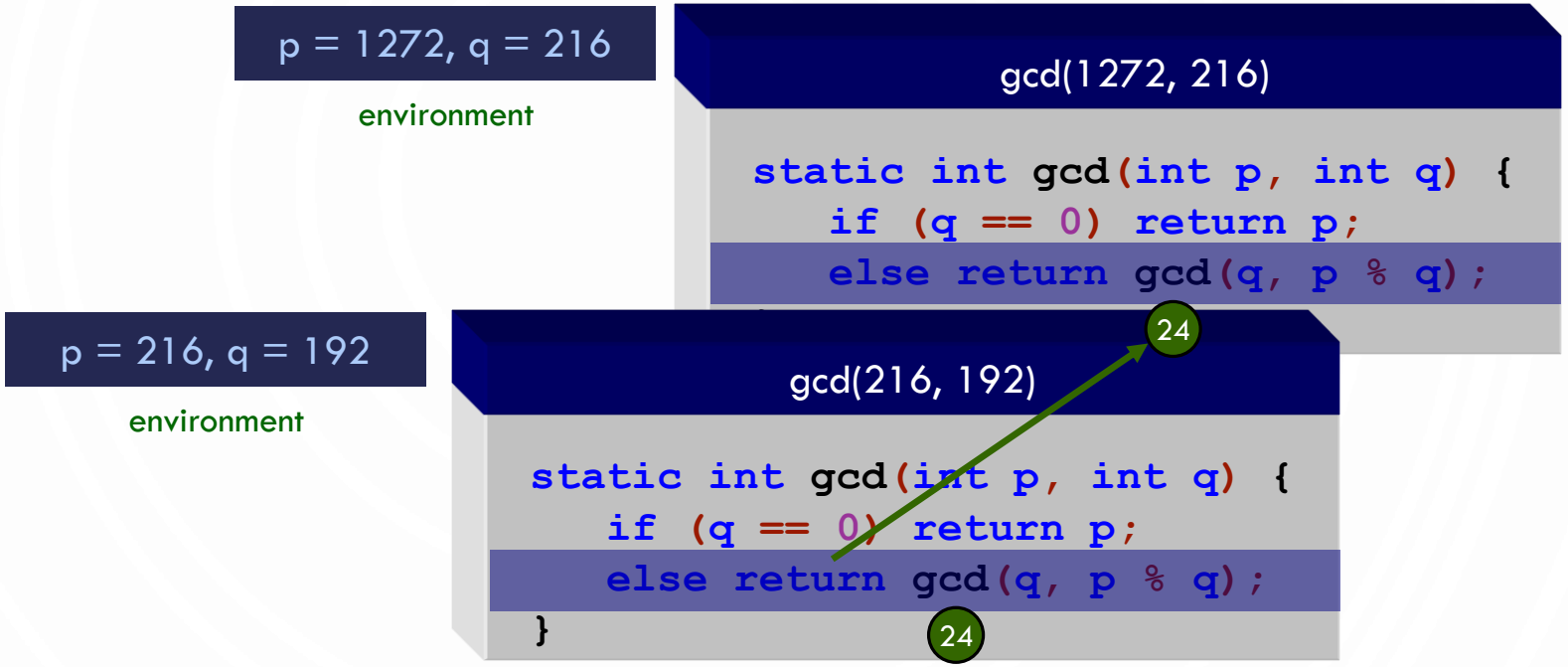
p = 216, q = 192

environment

gcd(216, 192)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
}
```

24


p = 1272, q = 216
environment
gcd(1272, 216)
static int gcd(int p, int q) {
if (q == 0) return p;
else return gcd(q, p % q);
p = 216, q = 192
environment
gcd(216, 192)
static int gcd(int p, int q) {
if (q == 0) return p;
else return gcd(q, p % q);
}
24
24

p = 1272, q = 216

environment

gcd(1272, 216)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
}
```

24

p = 1272, q = 216

environment

gcd(1272, 216)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
}
```

24

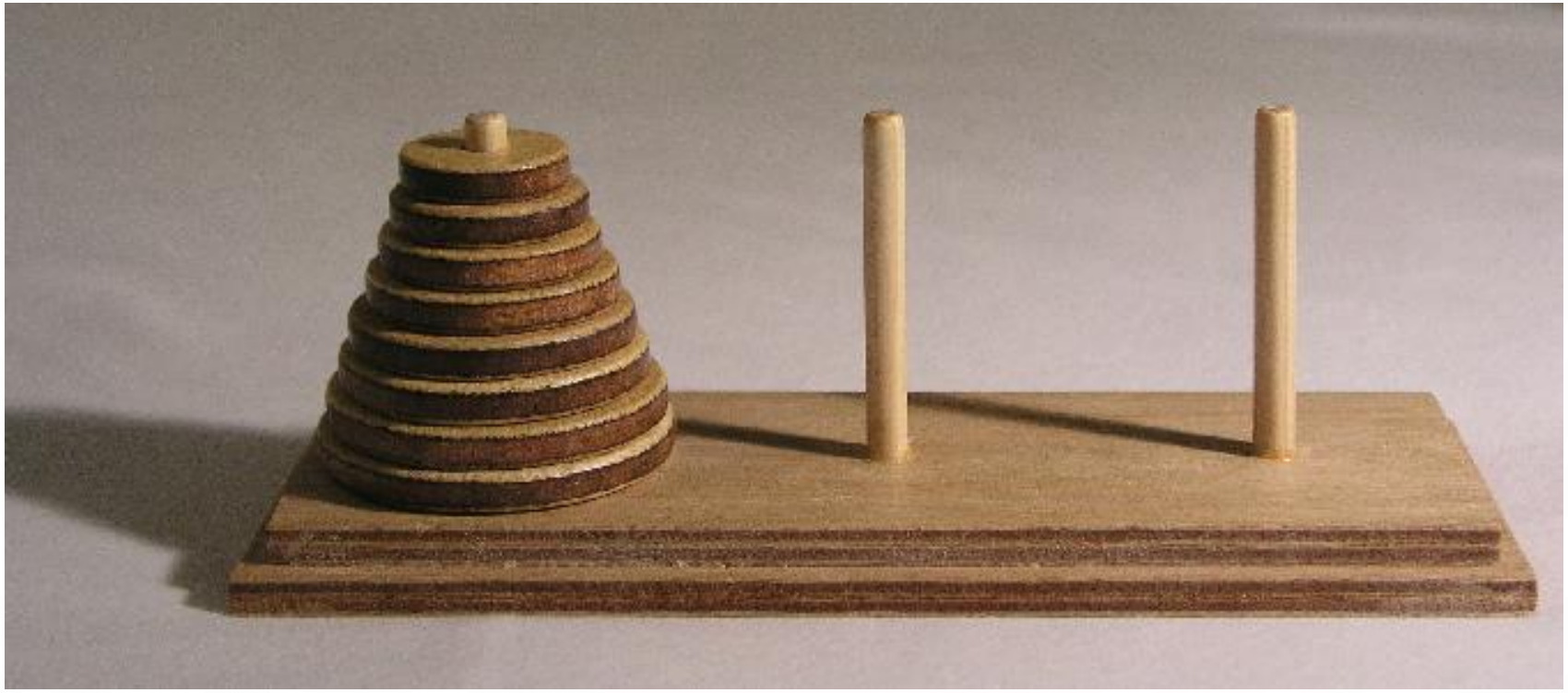
```
public class Euclid {
   public static int gcd(int p, int q) {
      if (q == 0) return p;
      else return gcd(q, p % q);
   }

   public static void main(String[] args) {
      int p = Integer.parseInt(args[0]);
      int q = Integer.parseInt(args[1]);
      System.out.println(gcd(p, q));
   }
}
```

24

```
% java Euclid 1272 216
24
```

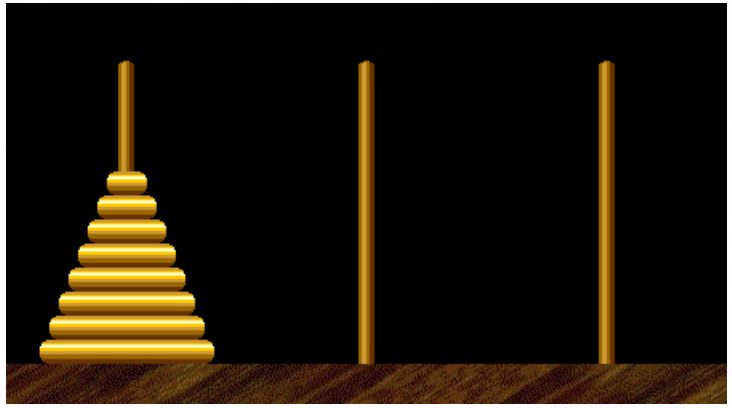
TOWERS OF HANOI



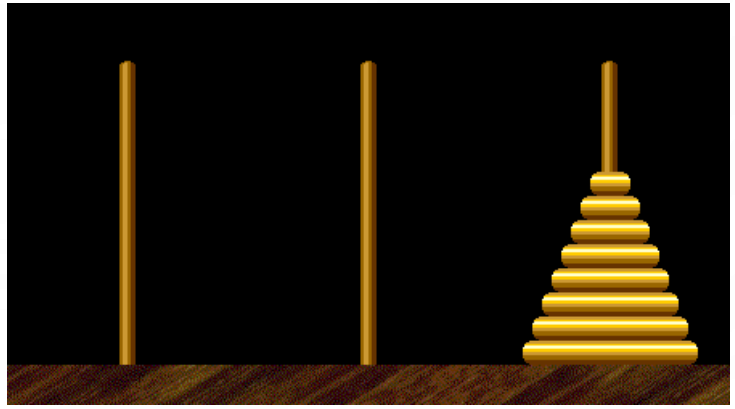
http://en.wikipedia.org/wiki/Image:Hanoiklein.jpg

PRACTICE
TOWERS OF HANOI

- Design recursive algorithm to move all the discs from the leftmost peg to the rightmost one.
  - Only one disc may be moved at a time.
  - A disc can be placed either on empty peg or on top of a larger disc.

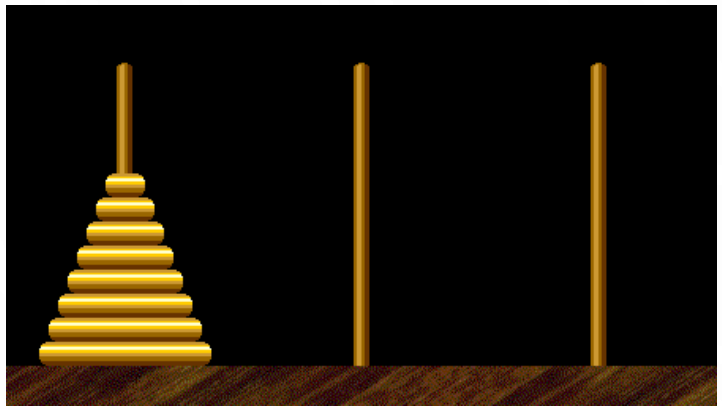


start

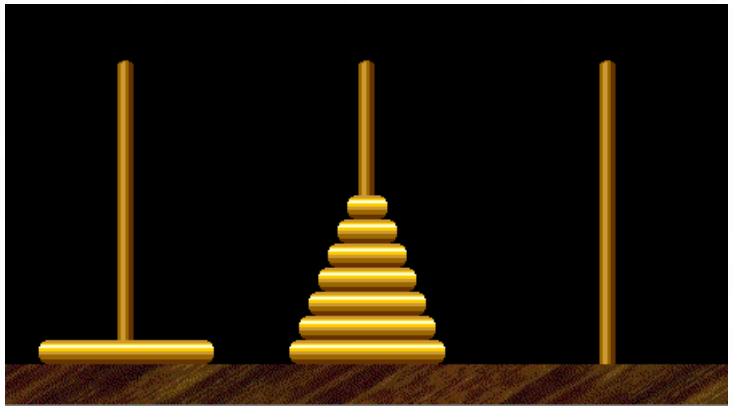


finish

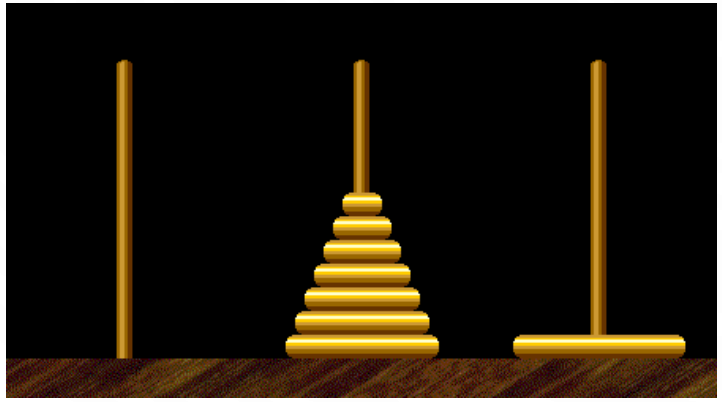
SOLUTION
TOWERS OF HANOI



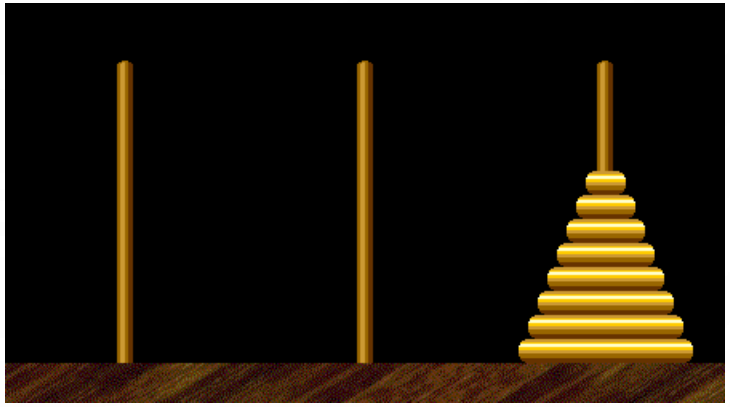
Move n-1 smallest discs right.



Move largest disc left.



Move n-1 smallest discs right.

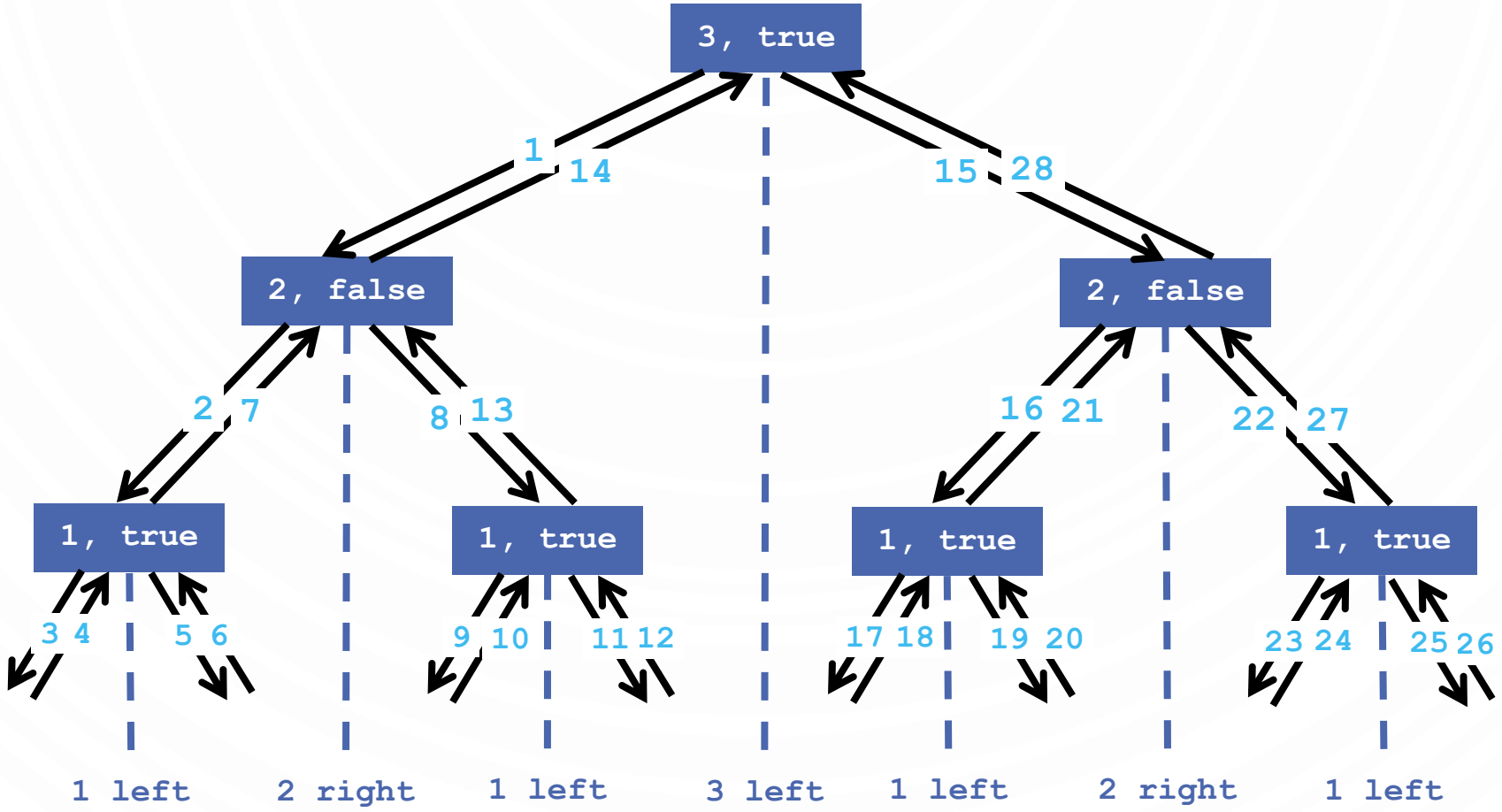


CODE
TOWERS OF HANOI

```
1.  public class TowersOfHanoi {
2.    public static void moves(int n, boolean left)
    {
        if (n == 0) return;
3.      moves(n-1, !left);
4.      if (left) System.out.println(n + " left");
5.      else      System.out.println(n + " right");
6.      moves(n-1, !left);
7.    }
8.
9.    public static void main(String[] args) {
10.     int N = Integer.parseInt(args[0]);
11.     moves(N, true);
12.   }
13. }
```

```
% java TowersOfHanoi 3
1 left
2 right
1 left
3 left
1 left
2 right
1 left
```

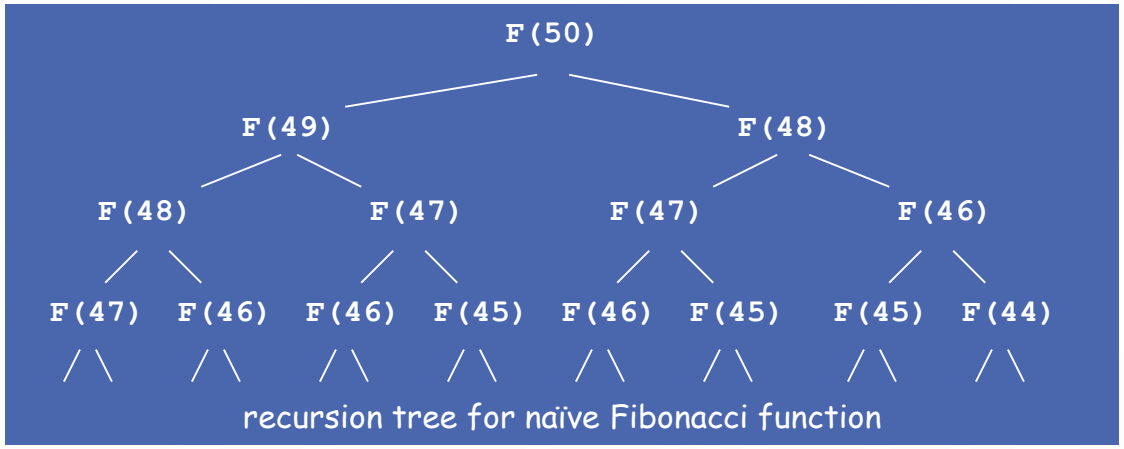
RECURSION TREE (FUNCTION TRACE)
TOWERS OF HANOI



DOWNSIDE OF RECURSION

- Recursion is not always efficient!
- Take for instance, the Fibonacci sequence

```
public static long F(int n) {
   if (n == 0) return 0;
   if (n == 1) return 1;
   return F(n-1) + F(n-2);
}
```



recursion tree for naïve Fibonacci function

- F(50) is called once.
- F(49) is called once.
- F(48) is called 2 times.
- F(47) is called 3 times.

...

- F(1) is called 12,586,269,025 times.

BEST PRACTICE
CONVERT RECURSIVE ALGORITHMS TO ITERATIVE ONES

- Try to do this whenever possible
- Example Fibonacci Sequence

```
1.  //This is an example conversion. You can be even more efficient!
2.  public static long F(int n) {
3.    if (n == 0) return 0;
4.    long[] F = new long[n+1]; //Allows reuse of computed values
5.    F[0] = 0;
6.    F[1] = 1;
7.    for (int i = 2; i <= n; i++) // Iterative means repetition until failure condition,
                                   // typically done with loops and not recursion
8.      F[i] = F[i-1] + F[i-2];
9.    return F[n];
10. }
```

SUMMARY

- How to write simple recursive programs?
  - Base case, reduction step.
- Trace the execution of a recursive program.
- Why learn recursion?
  - New mode of thinking.
  - Powerful programming tool