Recursion

Recursive Algorithm

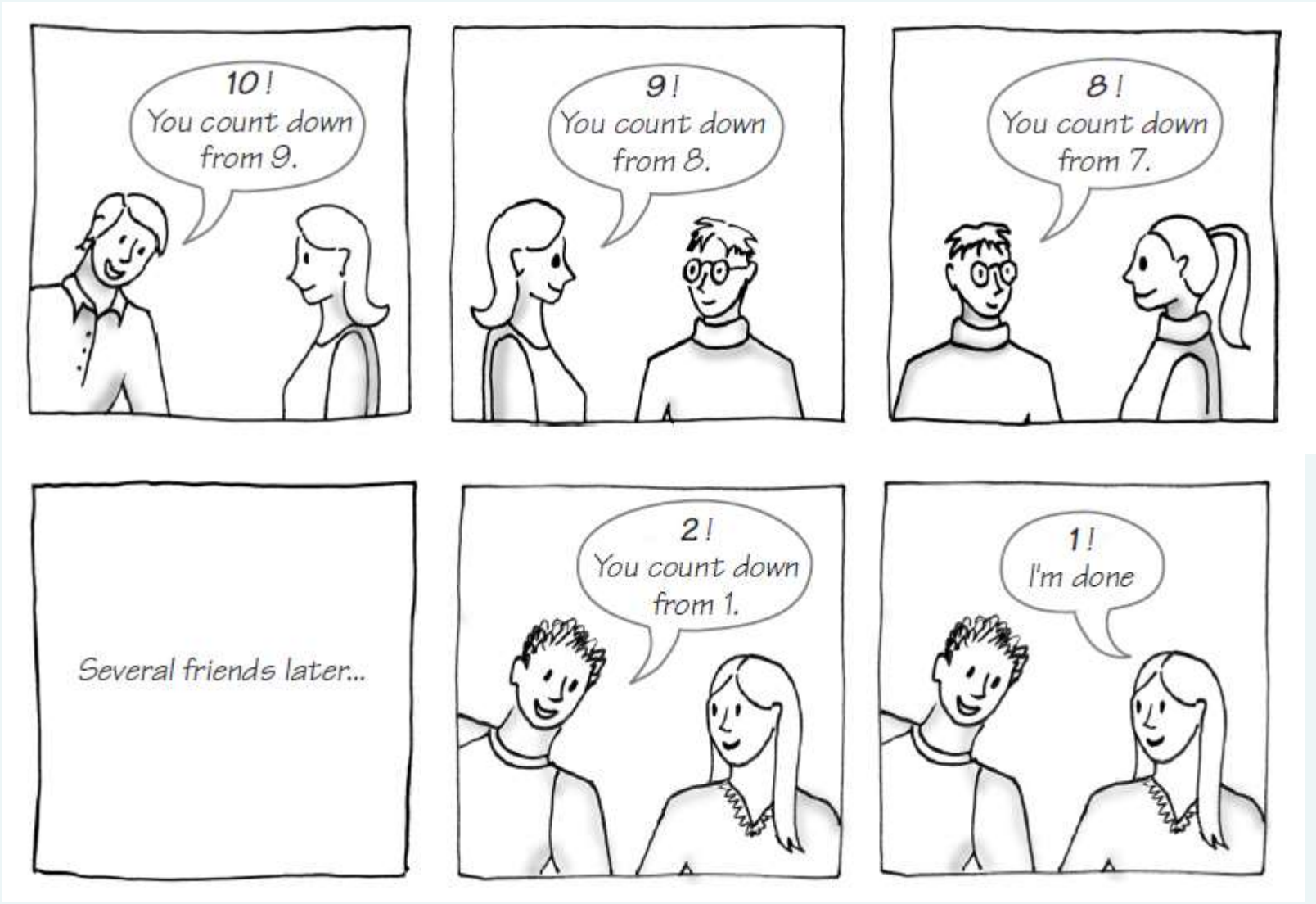
- Recursion
  - An extremely powerful problem-solving technique
  - Breaks a problem in smaller identical problems
  - Base Case

  - An alternative to iteration
    - An iterative solution involves loops

What Is Recursion?

- Consider hiring a contractor to build
  - He hires a subcontractor for a portion of the job
  - That subcontractor hires a sub-subcontractor to do a smaller portion of job
- The last sub-sub- … subcontractor finishes
  - Each one finishes and reports “done” up the line

Example: The Countdown



Characteristics of Recursion

All recursive methods have the following characteristics:

- One or more base cases (the simplest case) are used to stop recursion.
- Every recursive call reduces the original problem, bringing it increasingly closer to a base case until it becomes that case.

Example: The Countdown

$$\text{countdown}(1) = \text{print } 1$$

$$\text{countdown}(n) = \begin{cases} \text{print } n \\ \text{countdown}(n-1) \end{cases}$$

Recursive Java method to do countdown.

A Recursive Valued Method: The Factorial of n

- A recursive definition of factorial(n)

$$\text{factorial}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n * \text{factorial}(n-1) & \text{if } n > 0 \end{cases}$$

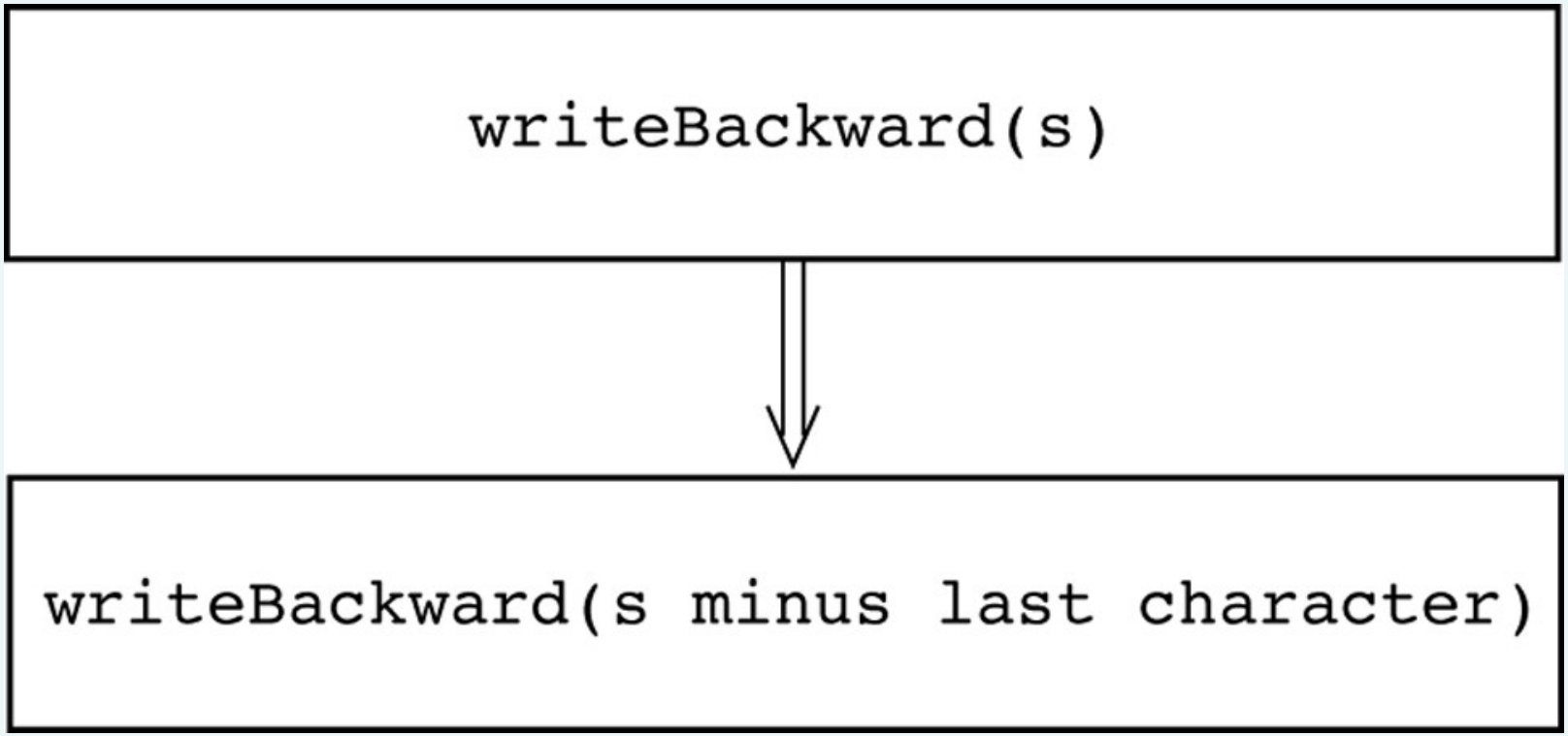
- A recurrence relation
  - A mathematical formula that generates the terms in a sequence from previous terms
  - Example

$$\text{factorial}(n) = n * [(n-1) * (n-2) * \ldots * 1]$$
$$= n * \text{factorial}(n-1)$$

A Recursive `void` Method: Writing a String Backward

- Problem
  - Given a string of characters, write it in reverse order
- Recursive solution
  - Each recursive step of the solution diminishes by 1 the length of the string to be written backward
  - Base case
    - Write the empty string backward

A Recursive `void` Method: Writing a String Backward



Multiplying Rabbits (The Fibonacci Sequence)

- Problem
  - How many pairs of rabbits are alive in month n?
- Recurrence relation

  fib(n) = fib(n-1) + fib(n-2)

`Fib series:`	**0**	**1**	**1**	**2**	**3**	**5**	**8**	**13**	**21**	**34**	**55**	**89**
`indices:`	0	1	2	3	4	5	6	7	8	9	10	11

Multiplying Rabbits (The Fibonacci Sequence)

- Base cases
  - fib(1), fib(0)
- Recursive definition

$$fib(n) = \begin{cases} \text{if } n==0 \rightarrow 0 \text{ , if } n==1 \rightarrow 1 \\ fib(n-1) + fib(n-2) \qquad \text{if } n > 2 \end{cases}$$

- Fibonacci sequence
  - The series of numbers fib(0), fib(1), fib(2), and so on

Palindrome

- a word, phrase, or sequence that reads the same backward as forward.
- e.g. madam or nurses run.
  - Iterate solution
  - Recursive solution

Binary Search

- A high-level binary search

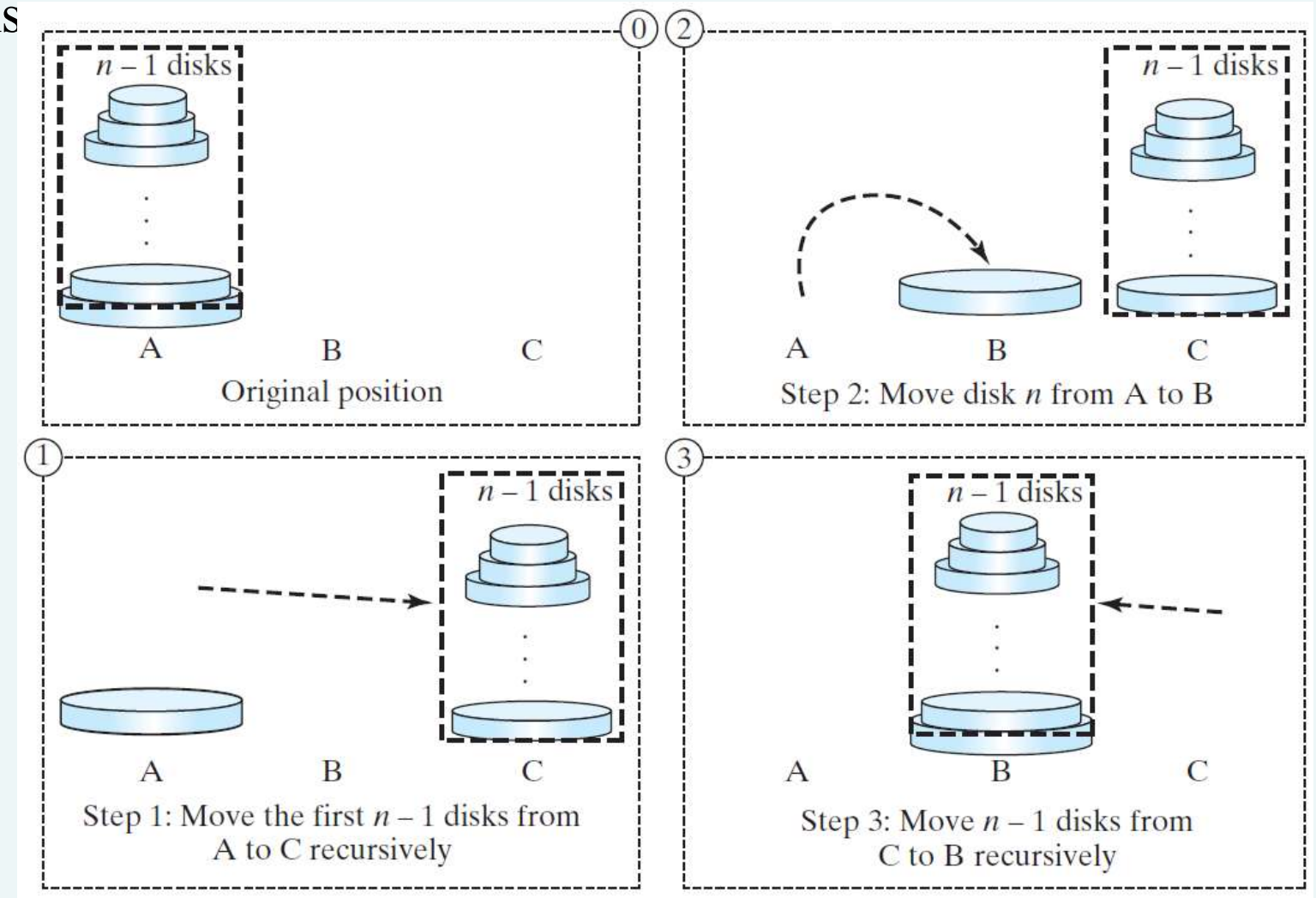
```
if (anArray is of size 1) {
 Determine if anArray's item is equal to value
}
else {
 Find the midpoint of anArray
 Determine which half of anArray contains value
 if (value is in the first half of anArray) {
   binarySearch (first half of anArray, value)
 }
 else {
   binarySearch(second half of anArray, value)
 } // end if
} // end if
```

Tower of Hanoi

- There are *n* disks labeled 1, 2, 3, . . ., *n*, and three towers labeled A, B, and C.
- No disk can be on top of a smaller disk at any time.
- All the disks are initially placed on tower A.
- Only one disk can be moved at a time, and it must be the top disk on the tower.

Solution to Tower of Hanoi

The Tower of Hanoi problem can be decomposed into three subproblems



The Towers of Hanoi

Tower oh Hanoi Animation

- Pseudocode solution

```
solveTowers(count, source, destination, spare)
 if (count is 1) {
   Move a disk directly from source to destination
 }
 else {
   solveTowers(count-1, source, spare, destination)
   solveTowers(1, source, destination, spare)
   solveTowers(count-1, spare, destination, source)
 } //end if
```

TowerOfHanoi.java

Exercise: GCD

gcd(2, 3) = 1

gcd(2, 10) = 2

gcd(25, 35) = 5

gcd(27, 18) = 9

gcd(m, n)

Approach 1: Brute-force, start from min(n, m) down to 1, to check if a number is common divisor for both m and n, if so, it is the greatest common divisor.

Approach 2: `Euclid's` algorithm

Approach 3: Recursive method

Approach 2: Euclid's algorithm

```
// Get absolute value of m and n;
t1 = Math.abs(m); t2 = Math.abs(n);
// r is the remainder of t1 divided by t2;
r = t1 % t2;
while (r != 0) {
  t1 = t2;
  t2 = r;
  r = t1 % t2;
}

// When r is 0, t2 is the greatest common
// divisor between t1 and t2
return t2;
```

Approach 3: `Recursive Method`

gcd(m, n) = n if m % n = 0;
gcd(m, n) = gcd(n, m % n); otherwise;