Northeastern University

Lecture 22: Recursion - 2

Prof. Chen-Hsiang (Jones) Yu, Ph.D.

College of Engineering

Materials are edited by Prof. Jones Yu from

Data Structures and Abstractions with Java, 5th edition. By Frank M. Carrano and Timothy M. Henry.
ISBN-13 978-0-13-483169-5

Apply Recursion to Learned Content

Recursively Processing an Array

Recursively Processing an Array

- Suppose that we have an array of integers and we want a method to display it.
- Of course, you can use `loop` to print all integers.
- Instead, how do you use recursion to achieve the same goal?

Recursively Processing an Array

```
/** Displays the integers in an array.
    @param array  An array of integers.
    @param first  The index of the first integer displayed.
    @param last   The index of the last integer displayed,
                  0 <= first <= last < array.length. */
public static void displayArray(int[] array, int first, int last)
```

Given definition of a recursive method to display array.

Recursively Processing an Array

```
public static void displayArray(int array[], int first, int last)
{
   System.out.print(array[first] + " ");
   if (first < last)
      displayArray(array, first + 1, last);
} // end displayArray
```

Starting with **`array[first]`**

Recursively Processing an Array

```
public static void displayArray(int array[], int first, int last)
{
   if (first <= last)
   {
      displayArray(array, first, last - 1);
      System.out.print(array[last] + " ");
   } // end if
} // end displayArray
```

Starting with **`array[last]`**

Recursively Processing an Array

```
int mid = (first + last) / 2;
```

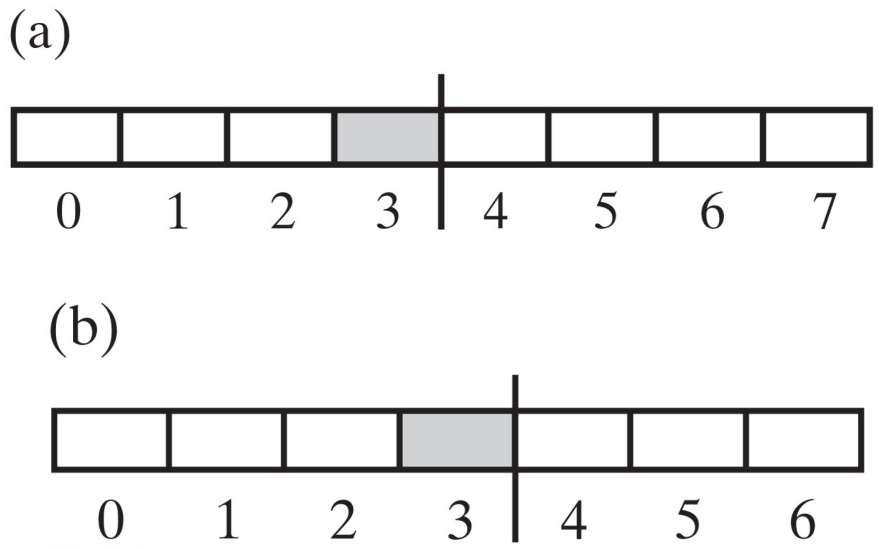

Figure 9-6: Two arrays with their middle elements within their left halves

Recursively Processing an Array

```
public static void displayArray(int array[], int first, int last)
{
   if (first == last)
      System.out.print(array[first] + " ");
   else
   {
      int mid = first + (last - first) / 2;
      displayArray(array, first, mid);
      displayArray(array, mid + 1, last);
   } // end if
} // end displayArray
```

Why?

Processing array from middle

Displaying a Bag

```
public void display()
{
   displayArray(0, numberOfEntries - 1);
} // end display

private void displayArray(int first, int last)
{
   System.out.println(bag[first]);
   if (first < last)
      displayArray(first + 1, last);
} // end displayArray
```

Recursive method that is part of an implementation of an ADT often is private

Recursively Processing a Linked Chain

Recursively Processing a Linked Chain

```
public void display()
{
   displayChain(firstNode);
} // end display

private void displayChain(Node nodeOne)
{
   if (nodeOne != null)
   {
      System.out.println(nodeOne.getData()); // Display data in first node
      displayChain(nodeOne.getNextNode());   // Display rest of chain
   } // end if
} // end displayChain
```

Display data in first node and recursively
display data in rest of chain

Recursively Processing a Linked Chain

```
public void displayBackward()
{
   displayChainBackward(firstNode);
} // end displayBackward

private void displayChainBackward(Node nodeOne)
{
   if (nodeOne != null)
   {
      displayChainBackward(nodeOne.getNextNode());
      System.out.println(nodeOne.getData());
   } // end if
} // end displayChainBackward
```

The order matters.

Displaying a chain backwards. Traversing chain of linked nodes in reverse order easier when done recursively.

Exercise (L22_E1)

- What is the output of this program?

```
public class Recursion {

    public int func(int n){
        int result;
        if (n == 1)
            return 1;
        result = func(n - 1);
        return result;
    }

    public static void main(String args[]) {
        recursion obj = new recursion() ;
        System.out.print(obj.func(5));
    }

}
```

Answer

```
1
```

A Simple Solution to a Difficult Problem

Simple Solution to a Difficult Problem

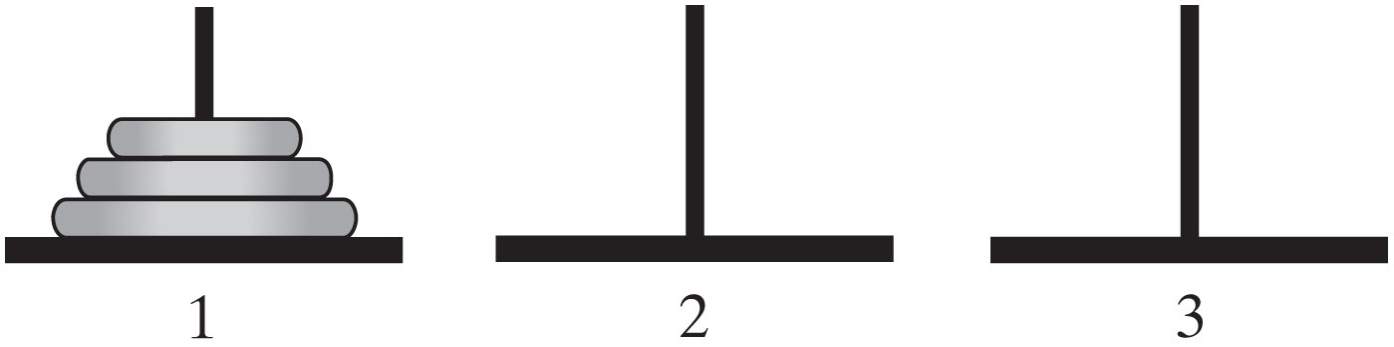

Figure 14-1: The initial configuration of the Towers of Hanoi for three disks.

Simple Solution to a Difficult Problem

- Rules:
  - » Move one disk at a time. Each disk moved must be topmost disk.
  - » No disk may rest on top of a disk smaller than itself.
  - » You can store disks on the second pole temporarily, as long as you observe the previous two rules.

Solutions

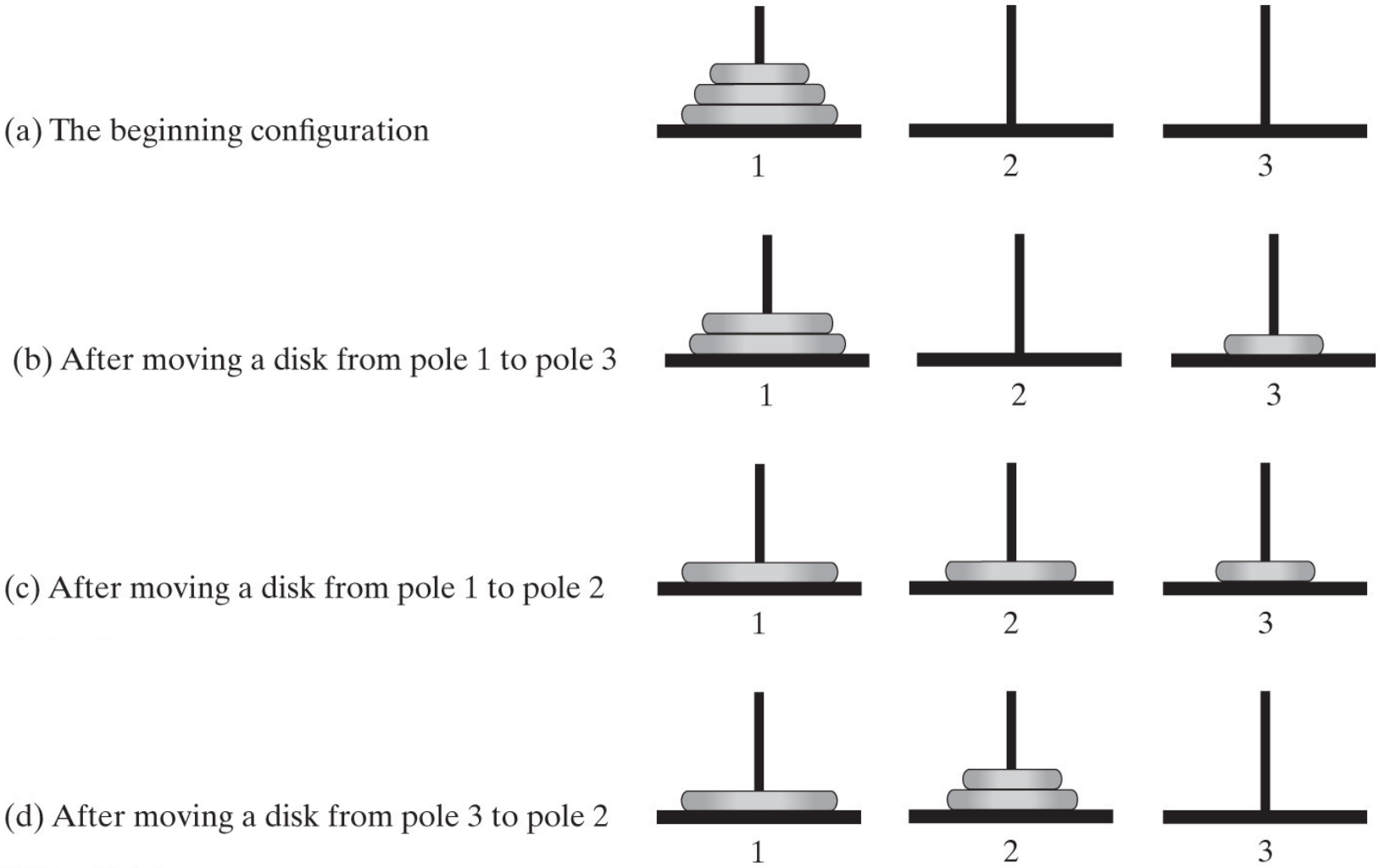

Figure 14-2: The sequence of moves for solving the Towers of Hanoi problem with three disks

Solutions

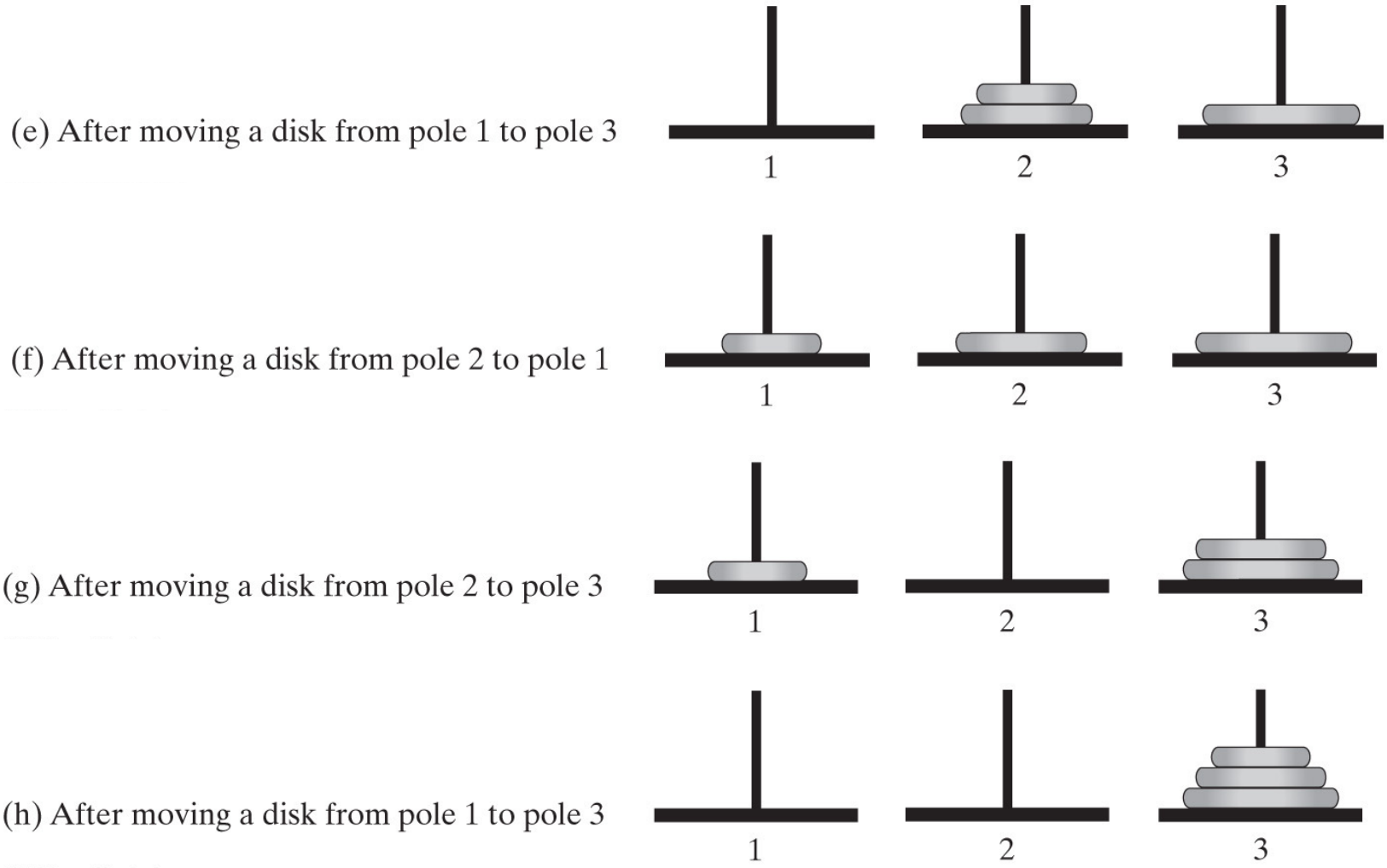

Figure 14-2: The sequence of moves for solving the Towers of Hanoi problem with three disks

A Smaller Problem

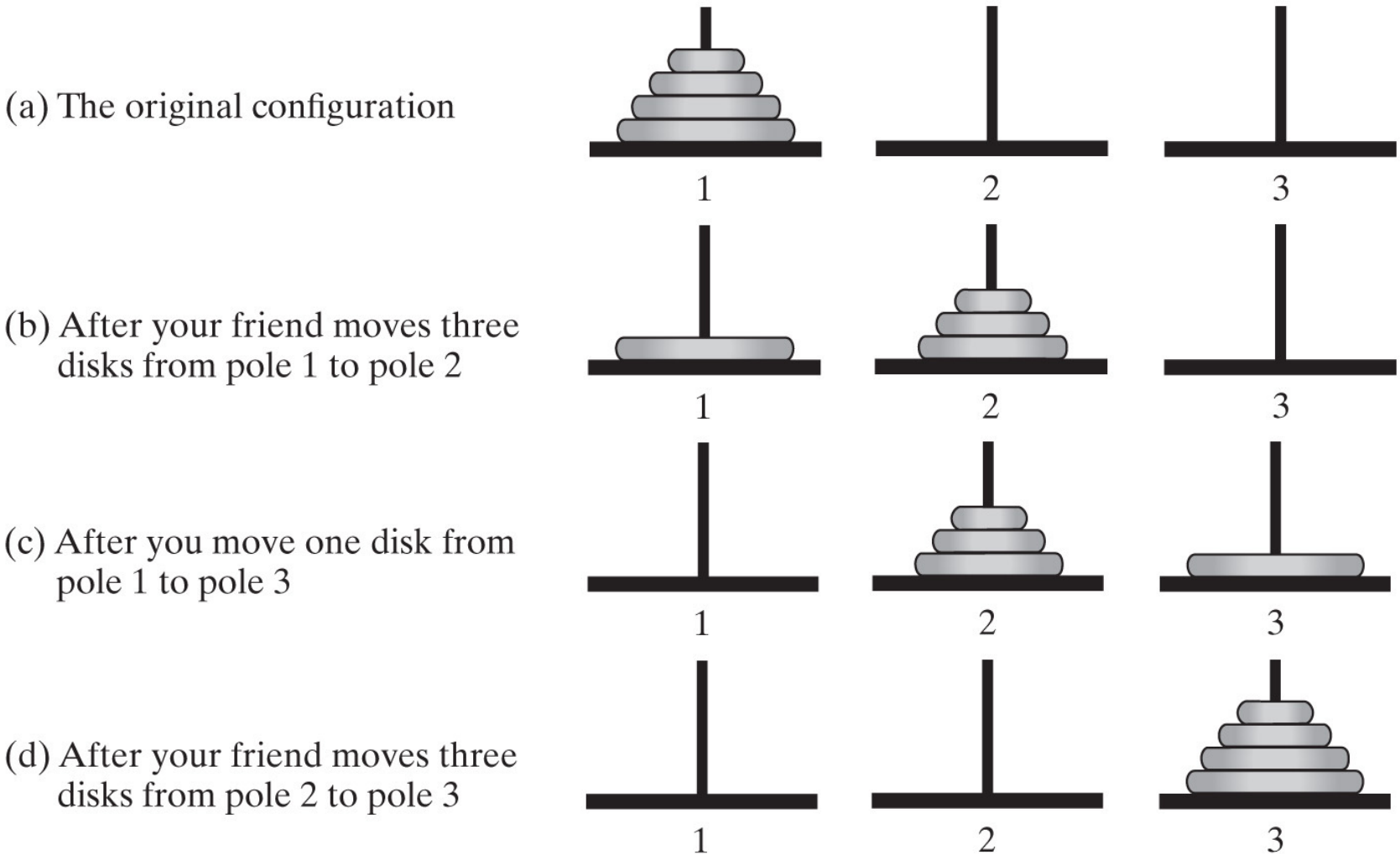

Figure 14-3: The smaller problems in a recursive solution for four disks

Solutions

Algorithm to move `numberOfDisks` disks from `startPole` to `endPole` using `tempPole` as a spare according to the rules of the Towers of Hanoi problem

```
if (numberOfDisks == 1)
   Move disk from startPole to endPole
else
{
   Move all but the bottom disk from startPole to tempPole
   Move disk from startPole to endPole
   Move all disks from tempPole to endPole
}
```

Recursive algorithm to solve any number of disks.

Note: for *n* disks, solution will be $2^n - 1$ moves

Exercise (L21_E2)

- Write a method that asks the user for integer input that is between 1 and 10, inclusively. If the input is out of range, the method should recursively ask the user to enter a new input value.

Answer

```
public static int getInput(Scanner keyboard)
{

    System.out.print("Please enter an integer between 1 and 10: ");
    int input = keyboard.nextInt();

    if ((input < 1) || (input > 10))
    {
        System.out.println(input + " does not lie between 1 and 10.");
        input = getInput(keyboard);
    } // end if
    return input;

} // end getInput
```