More Recursion

2025 Winter APS105: Computer Fundamentals
Jon Eyolfson

Lecture 25
1.0.0

A Recursive Function Calls Itself

We need two things:

1. a base case: a simple solution we know
2. a recursive step: reduces the problem to a smaller version of itself

Computing the Greatest Common Divisor (GCD)

The GCD of two integers `a` and `b`, is the largest integer `d` that is a divisor of both `a` and `b`

We'll assume all integers are positive and greater than 0

The Euclidean Algorithm for Finding the GCD

Find the largest common divisor, `d`, of integers `a` and `b`

Given: `a` $\geq$ `b`
  Replace `gcd(a, b)` with `gcd(b, a % b)`
    until `gcd(d, 0)`, where `d` is the GCD

We can write a recursive solution to this problem!

Finding the GCD in C

```
int gcd(int a, int b) {
    if (b == 0) {
        return a;
    }
    if (a >= b) {
        return gcd(b, a % b);
    }
    else {
        return gcd(b, a);
    }
}
```

For more practice, you could try to solve this using a loop instead

Can We Count to 5 Recursively?

Think about how we'd write this function

Counting from 1 to 5 Recursively

```c
#include <stdio.h>
#include <stdlib.h>

void count(int n) {
    if (n <= 0) {
        return;
    }
    printf("%d\n", n);
    count(n - 1);
}

int main(void) {
    count(5);
    return EXIT_SUCCESS;
}
```

What happens if we move `printf` to AFTER the recursive call?

Moving `printf` Counts from 5 to 1 Instead

```c
#include <stdio.h>
#include <stdlib.h>

void count(int n) {
    if (n <= 0) {
        return;
    }
    count(n - 1);
    printf("%d\n", n);
}

int main(void) {
    count(5);
    return EXIT_SUCCESS;
}
```

What About Computing the Sum of an Array Recursively?

We can use our same two rules for this as well!

Computing the Sum of An Array, Recursively

```
int sum(int *array, int arrayLength) {
    if (arrayLength == 0) {
        return 0;
    }
    else {
        return array[0] + sum(array + 1, arrayLength - 1);
    }
}
```

Maybe We Think of Another Solution

```
int sum(int *array, int arrayLength, int currentSum) {
    if (arrayLength == 0) {
        return currentSum;
    }
    else {
        return sum(array + 1, arrayLength - 1, array[0] + currentSum);
    }
}
```

Having an Extra Argument Can Be Confusing

Instead of:

```
int sum(int *array, int arrayLength, int currentSum);
```

It would be easier to use:

```
int sum(int *array, int arrayLength);
```

We can create a “helper” function that has all the arguments, and use it in our easier to use function

Another Sum Solution with a Helper Function

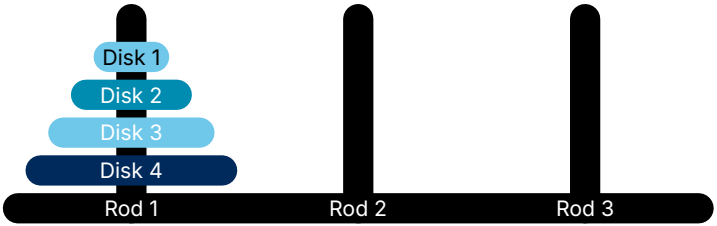
```
int sum_helper(int *array, int arrayLength, int currentSum) {
    if (arrayLength == 0) {
        return currentSum;
    }
    else {
        return sum_helper(array + 1, arrayLength - 1, array[0] + currentSum);
    }
}

int sum(int *array, int arrayLength) {
    return sum_helper(array, arrayLength, 0);
}
```

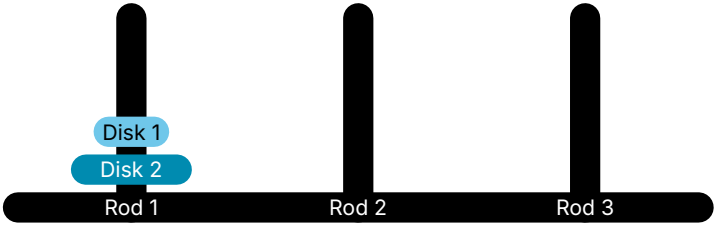
Can We Solve the Tower of Hanoi Recursively?

You want to move the tower of disks from rod 1 to 3 (peg numbers in white)

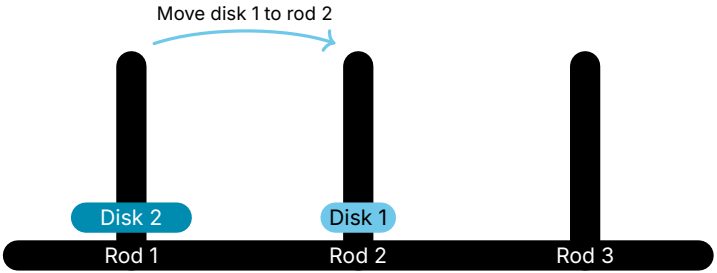
1. You can only move one disk at a time
2. You can move the top disk from a rod and place it at the top of another rod
3. You cannot place a larger disk on top of a smaller one



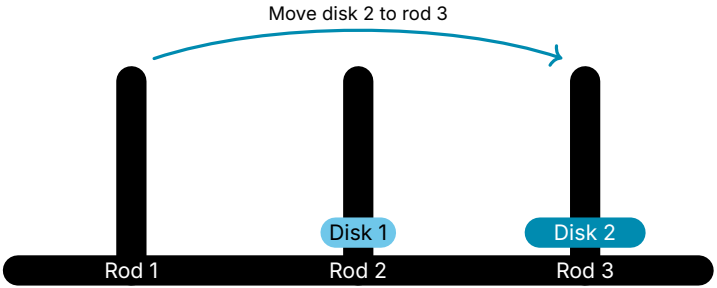
Let’s Solve the Tower of Hanoi with 2 Disks



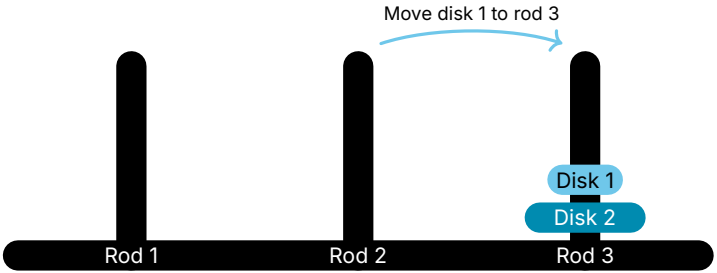
Let’s Solve the Tower of Hanoi with 2 Disks



Let’s Solve the Tower of Hanoi with 2 Disks

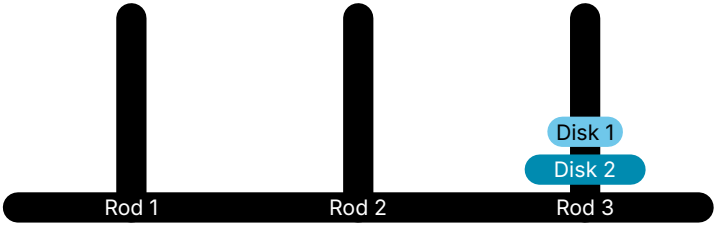


Let’s Solve the Tower of Hanoi with 2 Disks

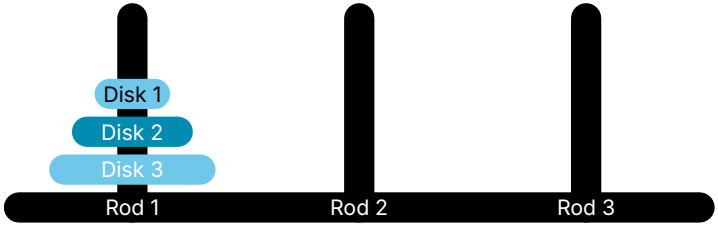


Let’s Solve the Tower of Hanoi with 2 Disks

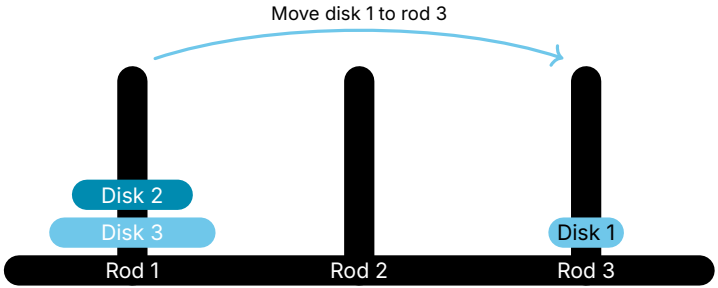
What happens if we move disk 1 to rod 3 first?



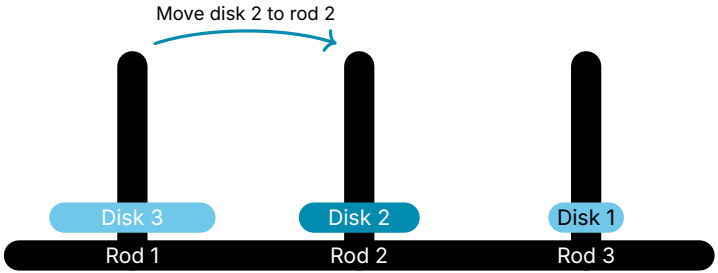
Let’s Solve the Tower of Hanoi with 3 Disks



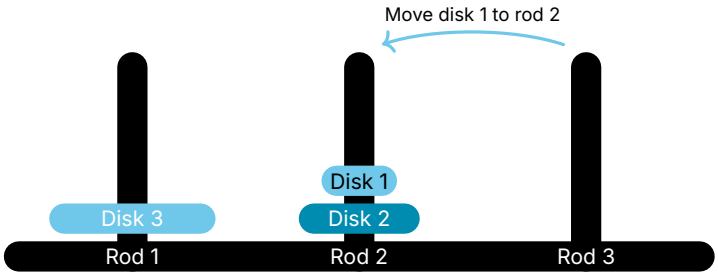
Let’s Solve the Tower of Hanoi with 3 Disks



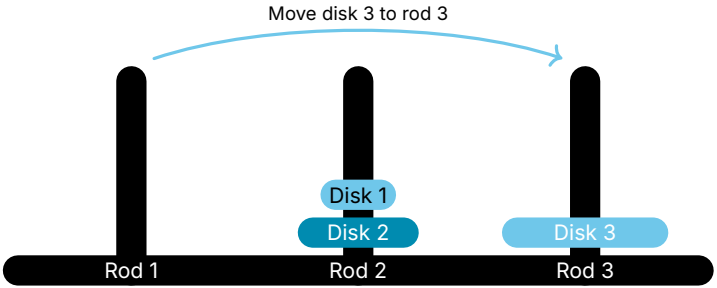
Let’s Solve the Tower of Hanoi with 3 Disks



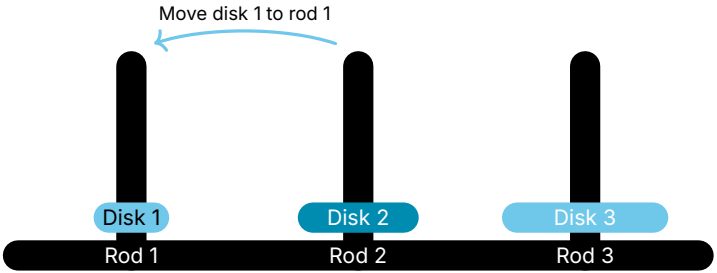
Let’s Solve the Tower of Hanoi with 3 Disks



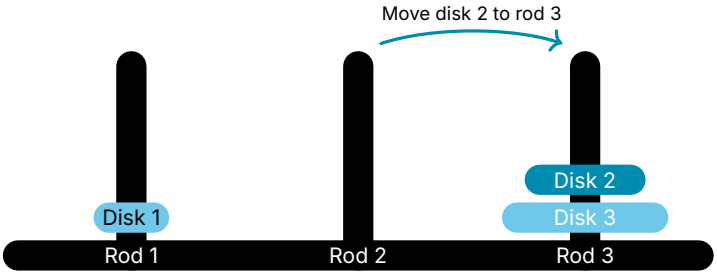
Let’s Solve the Tower of Hanoi with 3 Disks



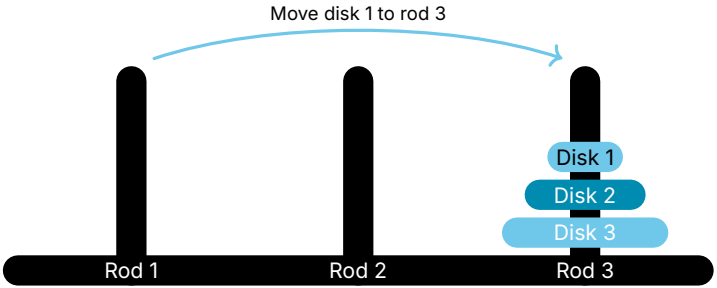
Let’s Solve the Tower of Hanoi with 3 Disks



Let’s Solve the Tower of Hanoi with 3 Disks

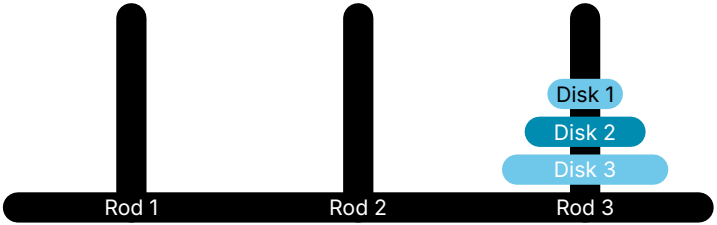


Let’s Solve the Tower of Hanoi with 3 Disks



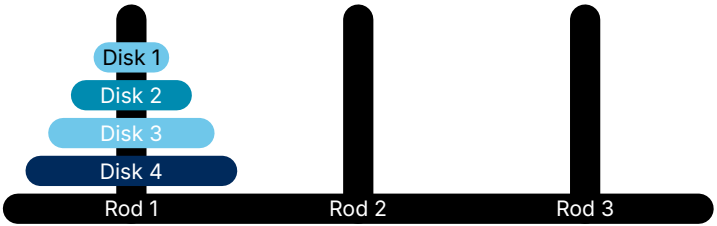
Let’s Solve the Tower of Hanoi with 3 Disks

We can solve this in 7 steps. Did you notice a recursive pattern?

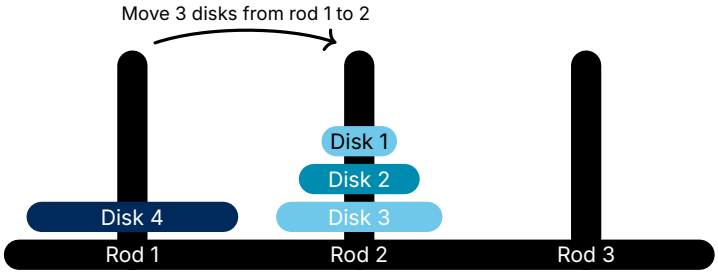


Discovering the Recursive Pattern to the Tower of Hanoi

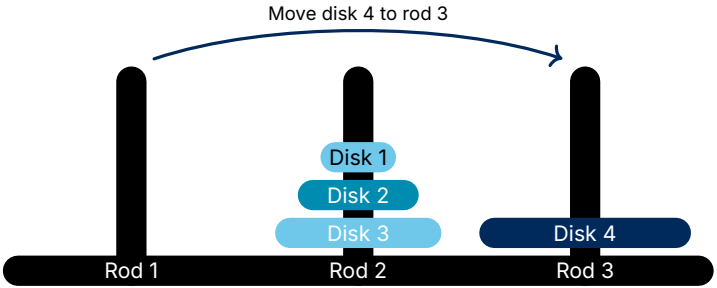
Problem: Move 4 disks from rod 1 to 3



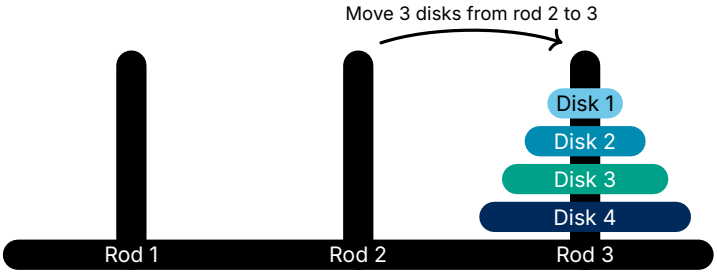
Discovering the Recursive Pattern to the Tower of Hanoi



Discovering the Recursive Pattern to the Tower of Hanoi



Discovering the Recursive Pattern to the Tower of Hanoi



We Can Solve the Tower of Hanoi with 2 Subproblems

Generalize the rods: from, to, and spare

We have a subproblem:
any smaller disk can go on top of the one we're solving for

To move n disks from **from** to **to** using **spare** as spare

1. Move n – 1 disks from **from** to **spare** using **to** as spare
2. Move disk n from **from** to **to**
3. Move n – 1 disks from **spare** to **to** using **from** as spare

What are we missing?

Our Tower of Hanoi Solution is Compact

```
int hanoi(int disks, int from_rod, int to_rod, int spare_rod) {
    /* Base case */
    if (disks == 0) {
        return 0;
    }
    /* Recursive steps */
    int steps = hanoi(disks - 1, from_rod, spare_rod, to_rod);
    printf("Move disk %d to rod %d\n", disks, to_rod);
    steps += 1;
    steps += hanoi(disks - 1, spare_rod, to_rod, from_rod);
    return steps;
}
```