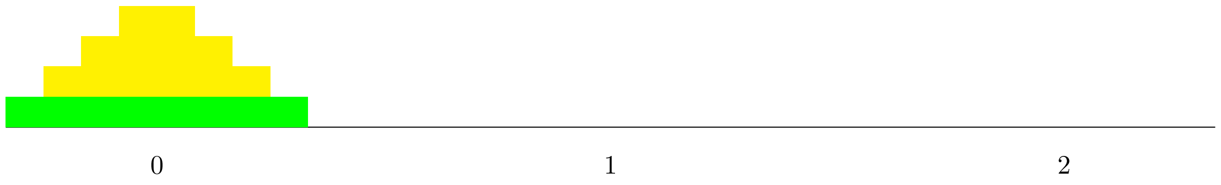
NOTES ON THE TOWERS OF HANOI

ERIC MARTIN

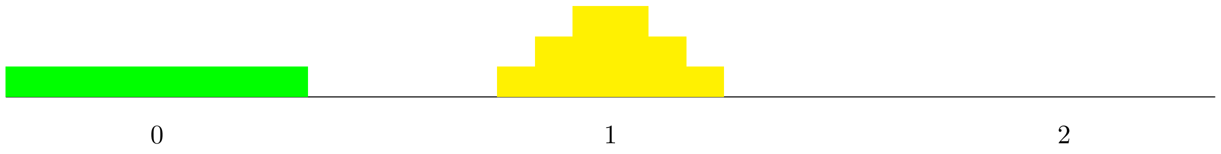
Let N be an integer at least equal to 1. A total of N disks all of different sizes, stacked at position 0, have to be moved to position 2, with position 1 available, thanks to a succession of moves each of which consists of removing the disk at the top of a stack and bringing it to the top of another (possibly empty) stack. At any stage of the game, no disk should be above a smaller disk; in particular, at the beginning of the game, all disks are stacked from largest to smallest.

It is necessary to eventually move the largest disk from position 0 to position 2. That is possible only when the $N-1$ smaller disks have been moved away from position 0 and stacked at position 1. Then, with the largest disk having found its final position at the base of position 2, those disks have to be moved away from position 1 and stacked at position 2. This is illustrated as follows for $N = 4$.

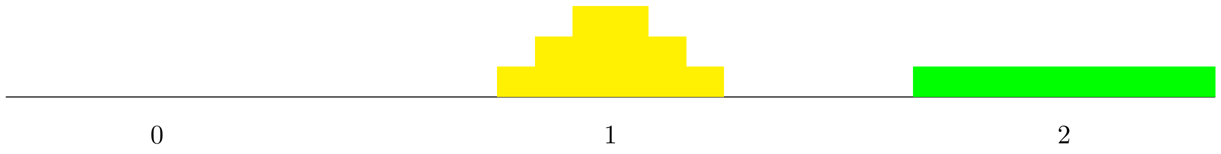
Initial configuration:



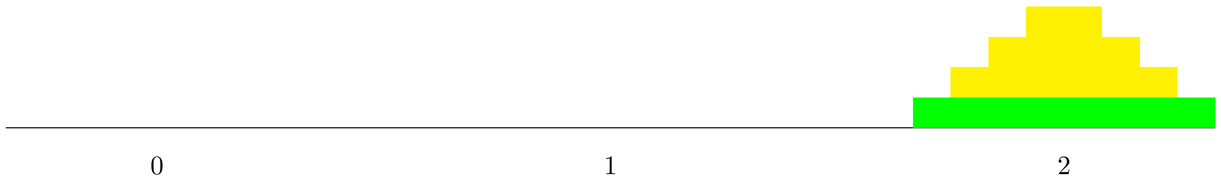
After the $N-1$ smaller disks have been moved from position 0 to position 1:



Largest disk moved from position 0 to position 2:



Final configuration, after the $N-1$ smaller disks have been moved from position 1 to position 2:



The recursive implementation immediately follows from that observation:

```
def move_towers(n, start_position, end_position, intermediate_position):
    if n == 1:
        print(f'Move smallest disk from {start_position} to {end_position}')
    else:
        move_towers(n - 1, start_position, intermediate_position, end_position)
        print(f'Move disk of size {n} from {start_position} to {end_position}')
        move_towers(n - 1, intermediate_position, end_position, start_position)
```

Date: Session 2, 2017.

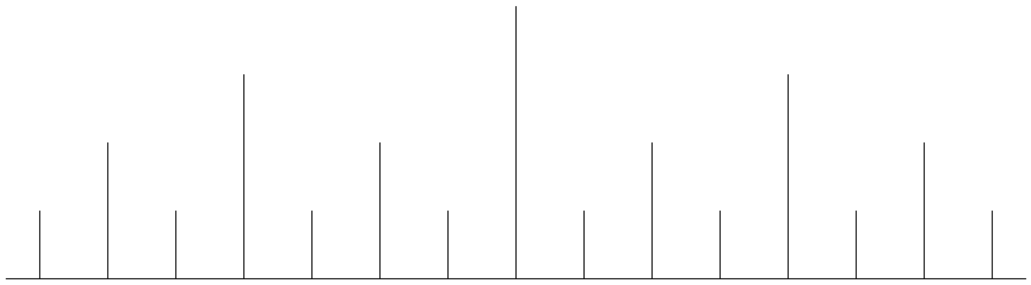
A call to `move_towers(4, 0, 2, 1)` yields this output:

```
Move smallest disk from 0 to 1
Move disk of size 2 from 0 to 2
Move smallest disk from 1 to 2
Move disk of size 3 from 0 to 1
Move smallest disk from 2 to 0
Move disk of size 2 from 2 to 1
Move smallest disk from 0 to 1
Move disk of size 4 from 0 to 2
Move smallest disk from 1 to 2
Move disk of size 2 from 1 to 0
Move smallest disk from 2 to 0
Move disk of size 3 from 1 to 2
Move smallest disk from 0 to 1
Move disk of size 2 from 0 to 2
Move smallest disk from 1 to 2
```

Also, that observation establishes that the output of the recursive implementation is not only a solution; it is in fact the only possible solution for that optimal number of moves, equal to $2^N - 1$, as shown by induction on N:

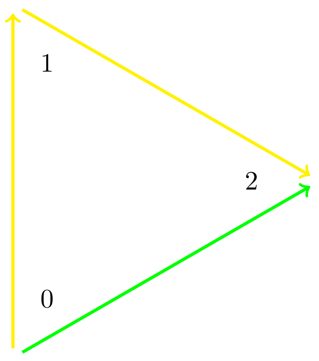
- If $N = 1$, then $2^N - 1 = 1$, and 1 move is necessary and sufficient indeed.
- If $2^N - 1$ moves are necessary and sufficient to move N disks, then $(2^N - 1) \times 2 + 1 = 2^{N+1} - 1$ moves are necessary and sufficient indeed to move $N + 1$ disks.

Moving the smaller $N - 1$ disks, then moving the largest disk, then moving the $N - 1$ smaller disks again to move the N disks, done recursively, shows that the Hanoi of towers puzzle is equivalent to that of putting ticks of N different sizes on a ruler, as illustrated next for $N = 4$:



This shows that every second move involves moving the smallest disk; the other moves are clearly imposed. So to convert the recursive implementation into an iterative implementation, it suffices to determine where to move the smallest disk every time it has to be moved.

Place the three positions, 0, 1 and 2, on a ring.



- Moving the N disks from position 0 to position 2 can be visualised as moving on the ring counterclockwise.
- Moving the $N - 1$ disks from position 0 to position 1 can be visualised as moving on the ring clockwise.
- Moving the $N - 1$ disks from position 1 to position 2 can be visualised as moving on the ring clockwise.

It follows that:

- the stack of N disks have to be moved from where they are to the next position counterclockwise, which if $N > 1$ involves
- twice moving the stack of the $N - 1$ smaller disks from where they are to the next position clockwise, which if $N > 2$ involves
- four times moving the stack of the $N - 2$ smaller disks from where they are to the next position counterclockwise, which if $N > 3$ involves
- eight times moving the stack of the $N - 3$ smaller disks from where they are to the next position clockwise
- ...

Hence:

- if N is even then the smallest disk always has to be moved to the next position clockwise, that is, from 0 to 1, from 1 to 2, and from 2 to 0;
- if N is odd then the smallest disk always has to be moved to the next position counterclockwise, that is, from 0 to 2, from 2 to 1, and from 1 to 0.

That justifies the following iterative implementation.

```
def move_towers(n, start_position, end_position, intermediate_position):
    smallest_disk_position = 0
    direction = 1 - n % 2 * 2
    stacks = list(range(n, 0, -1)), [], []
    for i in range(2 ** n - 1):
        if i % 2 == 0:
            new_smallest_disk_position = (smallest_disk_position + direction) % 3
            print(f'Move smallest disk from {smallest_disk_position} to {new_smallest_disk_position}')
            stacks[new_smallest_disk_position].append(stacks[smallest_disk_position].pop())
            smallest_disk_position = new_smallest_disk_position
        else:
            other_positions = tuple(set(range(3)) - {smallest_disk_position})
            if not stacks[other_positions[0]]:
                from_position, to_position = other_positions[1], other_positions[0]
            elif not stacks[other_positions[1]]:
                from_position, to_position = other_positions[0], other_positions[1]
            elif stacks[other_positions[0]][-1] < stacks[other_positions[1]][-1]:
                from_position, to_position = other_positions[0], other_positions[1]
            else:
                from_position, to_position = other_positions[1], other_positions[0]
            stacks[to_position].append(stacks[from_position].pop())
            print(f'Move disk of size {stacks[to_position][-1]} from {from_position} to {to_position}')
```