Towers of Hanoi (Induction & Recursion, Supplementary)

Table of contents

The puzzle 1
The proposition 2
Base case 2
The inductive step 2
Validating the algorithm: four disks 4
Exercise stub 6

Supplementary material reinforcing 2025-10-20-recursion[1] and python-recursion[2] — a formal induction proof that the Towers of Hanoi puzzle is always solvable, plus a fully validated worked trace.

The boards in the figures use colours that are not in csse1001-diagram-notation[3]: poles and base are tan bars, disks are pink bars (a different pink from the course's usual magenta), and disk width increases with disk number.

The puzzle

> Move one disk at a time. Bigger disks cannot be stacked on smaller disks. Prove the stack can be moved to the last pole.

The proof says **pole** and the code says **peg**. The two words mean the same thing here.

[1] courses/csse1001/2025-10-20-recursion.pdf
[2] courses/csse1001/python-recursion.pdf
[3] courses/csse1001/csse1001-diagram-notation.pdf

The proposition

The proposition is a biconditional with the two rules attached as a numbered list:

$P(n) \iff$ a stack of n-disks can be moved to an *arbitrary* pole while…

1. Moving one disk at a time.
2. Bigger disks never stacked on smaller disks.

The word *arbitrary* is doing real work: the claim is not "the stack can be moved to one particular pole", but "to whichever pole you name". That is what makes the hypothesis strong enough to be used **twice** in the step below — once to reach the spare pole, once to reach the target.

Base case

$P(0) \iff$ *zero* disks can be moved to an *arbitrary* pole while …

With zero disks the board is three empty poles. This is a **vacuous** case: with nothing to move, there is no move that could break either rule, so the conditions hold trivially.

The inductive step

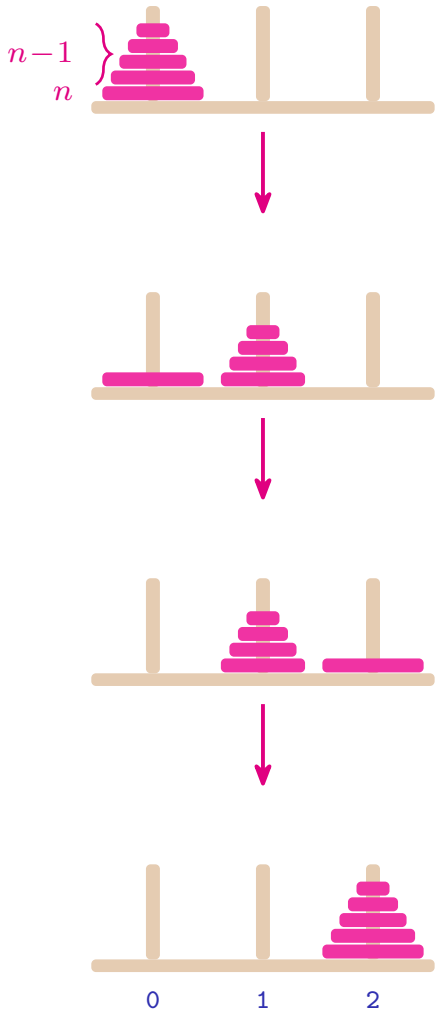
Assume $P(n-1)$ — that is, suppose we can move $n-1$ disks to any pole we like, obeying both rules. A stack of n disks then crosses the board in four states, and therefore $P(n)$. By the PMI, $\forall n; P(n)$.

Written out:

- **Base case** $P(0)$: vacuously true — there is nothing to move.
- **Inductive step**: assume $P(n-1)$. Then, for a stack of n disks:

  1. Move the top $n-1$ disks onto the spare pole (possible by the inductive hypothesis).
  2. Move the remaining, largest disk onto the target pole directly. This one move is legal precisely because step 1 uncovered it and emptied the target.
  3. Move the $n-1$ disks from the spare pole onto the target pole, on top of the largest disk (again possible by the inductive hypothesis, and legal because every disk in that stack is smaller than the one it lands on).

  This gives a valid sequence of moves for n disks, so $P(n)$ holds.
- **Conclusion**: by the Principle of Mathematical Induction, $P(n)$ holds for all n.



Start. All n disks on peg 0 — the state $P(n)$ asks us to move.

`hanoi(n-1, from_peg, off_peg)`

Move the top $n-1$ disks to the spare pole. Legal by the hypothesis $P(n-1)$.

Disk n is uncovered, and the target pole is empty.

`[(n, to_peg)]`

One legal move: nothing sits on disk n, and nothing is on peg 2 for it to land on.

Disk n is home, and never moves again.

`hanoi(n-1, off_peg, to_peg)`

$P(n-1)$ a second time. Every disk in that stack is smaller than disk n, so landing on it breaks no rule.

Therefore $P(n)$. By the PMI, $\forall n\,;\, P(n)$.

Figure 1: The inductive step as four board states (tan poles and base, pink disks, widths increasing with disk number). Five disks stand in for a general n. The pegs are numbered 0/1/2. The magenta brace names the top n-1 disks and the bottom disk n. The three monospace code fragments are the three terms of the return statement of hanoi, with num_disks shortened to n. Nothing about the puzzle forces the spare pole to be the middle one — that is simply the arrangement drawn here, and it matches hanoi(4, 0, 2).

This is exactly the recursive structure of the `hanoi` function in 2025-10-20-recursion[4]: **the inductive hypothesis *is* the recursive call**, and the three steps above are the three terms of that function's single `return` statement, in order. The proof establishes only that a solution *exists*; it says nothing about the solution being the shortest one.

Validating the algorithm: four disks

The worked example is the 15-move solution for four disks, replayed one board at a time. The move list in full:

```
(1,1),(2,2),(1,2),(3,1),(1,0),(2,1),(1,1),(4,2),(1,2),(2,0),(1,0),
(3,2),(1,1),(2,2),(1,2)
```

A move is written `(disk, destination_peg)`. Disks are labelled 1 through n and pegs are labelled 0, 1, and 2. The **order inside the pair** is that the first entry is the disk and the second is where it lands, and disk `1` is the *smallest*. Both are fixed by the code, which appends exactly `[(num_disks, to_peg)]` for each move, and confirmed by reading the boards.

Grouping the same 15 moves by the structure of Figure 1 — 7 moves to clear disks 1–3 onto the spare peg, 1 move for disk 4, then 7 moves to bring disks 1–3 back on top:

```
(1,1) (2,2) (1,2) (3,1) (1,0) (2,1) (1,1)
(4,2)
(1,2) (2,0) (1,0) (3,2) (1,1) (2,2) (1,2)
```

That grouping is exactly what `hanoi(4, 0, 2)` from 2025-10-20-recursion[5] returns, as three concatenated pieces:

```
>>> hanoi(4, 0, 2)
[(1, 1), (2, 2), (1, 2), (3, 1), (1, 0), (2, 1), (1, 1),
 (4, 2),
 (1, 2), (2, 0), (1, 0), (3, 2), (1, 1), (2, 2), (1, 2)]
```

Two details worth reading off Figure 2. First, **the largest disk moves exactly once** (move 8), and once it is placed nothing above it ever needs to shift it again — the pivot the whole proof turns on. Second, **the spare peg is not fixed**: the outer call uses peg 1 as its spare, but the nested `hanoi(3, 0, 1)` uses peg 2, which is why disks appear on the target peg long before disk 4 arrives there. That is `off_peg = 3 - from_peg - to_peg` doing its work at every level.

[4] courses/csse1001/2025-10-20-recursion.pdf
[5] courses/csse1001/2025-10-20-recursion.pdf

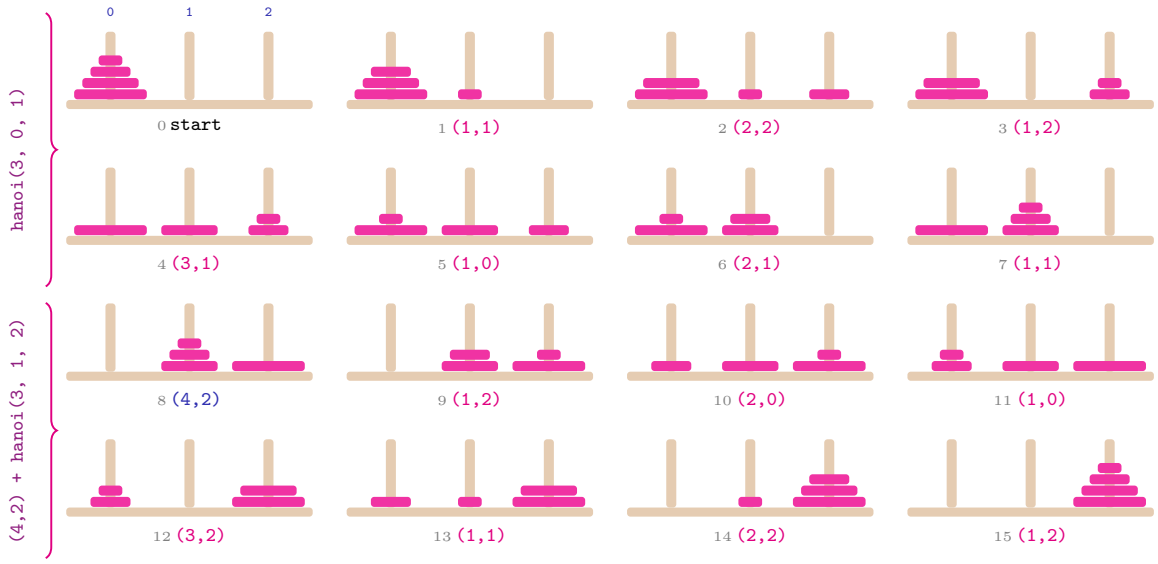


Figure 2: All sixteen board states of the four-disk solution, in reading order. Each panel is captioned with the move that produced it: a small grey move number and the move itself in magenta. The peg numbers are on the first board. Move 8 is in blue — the single move of the largest disk, the pivot of the whole recursion. The two magenta braces name the calls each half of the trace comes from. The split is exact: the eight boards above it are the start state plus the seven moves of hanoi(3, 0, 1), and the eight below are (4,2) followed by the seven moves of hanoi(3, 1, 2).

How many moves

The count falls straight out of the code's shape: one call to `hanoi(n, ...)` emits whatever `hanoi(n-1, ...)` emits, twice, plus one move of its own, and `hanoi(0, ...)` emits none.

$$T(0) = 0, \qquad T(n) = 2\,T(n-1) + 1 \quad \Longrightarrow \quad T(n) = 2^n - 1$$

So four disks take $2^4 - 1 = 15$ moves, which is what the trace above shows, and the cost is exponential in the number of disks — see algorithmic-complexity[6] for that vocabulary.

Exercise stub

The implementation is left as an exercise, starting from only the signature and docstring:

```
def hanoi(n: int, from_peg: int, target_peg: int) -> list[tuple[int, int]]:
    """
    Solve towers of hanoi problem for <n> disks from <from_peg> to
    <target_peg>

    Disks are labelled 1 through n.

    Pegs are labelled 0, 1, and 2.
    """
```

2025-10-20-recursion[7] (Lecture 12A) has a complete implementation, with slightly different naming — `num_disks`/`to_peg` rather than `n`/`target_peg` — and a return annotation of `list[tuple[int]]` where this stub writes the more accurate `list[tuple[int, int]]`, since each move really is a pair. It also has the `off_peg = 3 - from_peg - to_peg` trick that picks the spare peg.

> Related: MATH1061's welcome & intro problems[8] previews the same puzzle (and the $2^n - 1$ minimum move count) as motivation for MATH1061's own later induction/recursion content — a nice example of the same idea showing up in two different courses.

[6] courses/csse1001/algorithmic-complexity.pdf
[7] courses/csse1001/2025-10-20-recursion.pdf
[8] courses/math1061/2026-07-27-welcome-and-intro-problems.pdf