Tower of Hanoi: Recursion and Algorithm Design

Joshua Terranova

May 19, 2025

1 Introduction: The Tower of Hanoi Puzzle

The Tower of Hanoi is not only a puzzle but also a profound representation of recursive thinking, often introduced in early computer science education to illustrate the concept of divide-and-conquer. The puzzle is set up with three rods—commonly labeled A (source), B (auxiliary), and C (destination)—and a set of n disks of different sizes which can slide onto any rod. The initial configuration places all n disks on rod A in decreasing size from bottom to top. The ultimate goal is to move the entire stack to rod C under strict rules that simulate real-world constraints such as physical stacking and order preservation.

Solving the Tower of Hanoi is not merely about moving pieces but about understanding how problems can be broken down into smaller, more manageable subproblems. Each time we attempt to move a disk, we find ourselves faced with a smaller Tower of Hanoi problem that must be resolved first. This inherent recursion is what makes the puzzle such a powerful teaching tool.

The number of moves required to solve the Tower of Hanoi problem grows rapidly as the number of disks increases. For just 3 disks, the solution requires 7 moves; for 4 disks, 15 moves; and for 64 disks—the legendary version said to be operated by monks in a temple—it would take over 580 billion years if one move were made every second. This rapid growth highlights the limitations of recursive algorithms and the importance of analyzing their efficiency.

The Tower of Hanoi is a classic mathematical puzzle attributed to the French mathematician *Édouard Lucas* in 1883. The puzzle consists of three vertical rods and a number of disks of different diameters, all initially stacked in ascending order (smallest at the top, largest at the bottom) on one of the rods. The goal of the puzzle is to move the entire stack to another rod, obeying the following rules:

- Only one disk can be moved at a time.
- Each move involves taking the upper disk from one of the stacks and placing it on top of another stack or an empty rod.
- No disk may be placed on top of a smaller disk.

Despite its simple rules, the Tower of Hanoi exhibits a rich recursive structure and is commonly used in the study of algorithm design, recursion, and mathematical induction. The puzzle has applications in computer science, particularly in the analysis of recursive algorithms and stack-based problem solving.

2 Derivation of the Recursive Formula

Let $T(n)$ denote the minimum number of moves required to transfer n disks from the source peg to the destination peg under the given rules. We can approach the problem recursively:

1. Move the top $n-1$ disks to an auxiliary peg. This requires $T(n-1)$ moves.
2. Move the largest (bottom) disk directly to the destination peg. This is 1 move.
3. Move the $n-1$ disks from the auxiliary peg to the destination peg on top of the largest disk. This again takes $T(n-1)$ moves.

Combining these steps, we get the recurrence:

$$T(n) = 2T(n-1) + 1$$

Base Case

When $n = 1$, the smallest case, we simply move the disk directly from the source peg to the destination peg. Hence,

$$T(1) = 1$$

Recursive Case

Assuming $T(n-1)$ holds, we can construct the solution for $T(n)$ using the recurrence relation above.

Closed-form Derivation

We conjecture a closed-form solution:

$$T(n) = 2^n - 1$$

This formula suggests that the number of moves doubles with each additional disk (minus one), consistent with exponential growth.

Proof by Mathematical Induction

Base Case: $n = 1$

$$T(1) = 2^1 - 1 = 1$$

This matches the recurrence base case.

Inductive Hypothesis: Assume $T(k) = 2^k - 1$ holds for some $k \geq 1$.

Inductive Step: Prove $T(k+1) = 2^{k+1} - 1$

$$\begin{aligned} T(k+1) &= 2T(k) + 1 \\ &= 2(2^k - 1) + 1 \\ &= 2^{k+1} - 2 + 1 \\ &= 2^{k+1} - 1 \end{aligned}$$

Thus, by mathematical induction, $T(n) = 2^n - 1$ for all $n \geq 1$.

3 Pseudocode and Recursive Structure

Pseudocode of Recursive Solution:

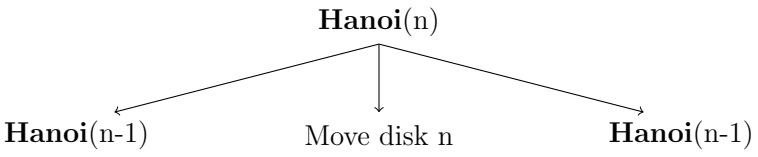
Listing 1: Tower of Hanoi Pseudocode

```
Algorithm Hanoi(n, source, target, auxiliary):
    if n == 1:
        print("Move disk 1 from", source, "to", target)
    else:
        Hanoi(n-1, source, auxiliary, target)
        print("Move disk", n, "from", source, "to", target)
        Hanoi(n-1, auxiliary, target, source)
```

This recursive algorithm effectively breaks down the problem into smaller subproblems, always moving $n-1$ disks before and after moving the largest disk.

This recursive process naturally forms a binary recursion tree, where each node represents a call to the Hanoi function with a specific number of disks. Each non-leaf node branches into two recursive calls: one for moving the top $n-1$ disks to the auxiliary peg, and another for moving them from the auxiliary to the target peg after the largest disk is moved. The depth of the recursion tree is n, and the total number of calls is $2^n - 1$, which matches the number of moves required.

Visual Diagram:



4 Worked-Out Example for $n = 4$

Here is a step-by-step move list for solving the Tower of Hanoi with 4 disks:

1. Move disk 1 from A to C
2. Move disk 2 from A to B
3. Move disk 1 from C to B
4. Move disk 3 from A to C
5. Move disk 1 from B to A
6. Move disk 2 from B to C
7. Move disk 1 from A to C
8. Move disk 4 from A to B
9. Move disk 1 from C to B
10. Move disk 2 from C to A
11. Move disk 1 from B to A
12. Move disk 3 from C to B
13. Move disk 1 from A to C
14. Move disk 2 from A to B
15. Move disk 1 from C to B

This completes the puzzle in exactly $2^4 - 1 = 15$ moves.

5 Proof of Correctness

We prove the algorithm is correct using structural induction.

Base Case

For $n = 1$, the algorithm performs one move from source to target, satisfying the requirement.

Inductive Step

Assume correctness for $n = k$: the algorithm correctly moves k disks according to the rules.

For $n = k + 1$, the algorithm:

- Recursively moves k disks to auxiliary peg (by assumption, this is valid).
- Moves the $k + 1$th (largest) disk directly to the target peg.
- Recursively moves the k disks from auxiliary to target.

Each of these operations respects the problem constraints. Hence, the algorithm correctly solves the puzzle for all n.

6 Time Complexity and Algorithm Efficiency

Recurrence and Solution

The recurrence relation:

$$T(n) = 2T(n-1) + 1$$

expands as:

$$T(n) = 2^n - 1$$

This yields a time complexity of:

$$T(n) \in \mathcal{O}(2^n)$$

The algorithm has exponential growth, making it infeasible for large n (e.g., $T(64) > 10^{19}$).

Comparison to Fibonacci Algorithm

The naive Fibonacci algorithm is defined as:

$$F(n) = F(n-1) + F(n-2)$$

This algorithm recalculates subproblems multiple times, leading to inefficiency.

In contrast, the Tower of Hanoi algorithm does not recompute subproblems; each recursive call performs unique work. Therefore, while both have exponential time complexity, Hanoi is more efficient in practice and lends itself to easier optimization or iteration.

7 Interactive Code Implementation with ASCII Visualization

The following Python 3 program solves the Tower of Hanoi recursively, accepts user input, and displays peg states using ASCII graphics.

Listing 2: Tower of Hanoi with ASCII Display

```
def print_towers(towers):
    max_height = max(len(peg) for peg in towers.values())
    for level in reversed(range(max_height)):
        for peg in ['A', 'B', 'C']:
            if level < len(towers[peg]):
                print(f" {towers[peg][level]} ", end="\t")
            else:
                print(" | ", end="\t")
        print()
    print(" A \t B \t C ")
    print("-" * 24)

def hanoi(n, source, target, auxiliary, towers):
    if n == 1:
        disk = towers[source].pop()
        towers[target].append(disk)
        print(f"Move disk 1 from {source} to {target}")
        print_towers(towers)
    else:
        hanoi(n - 1, source, auxiliary, target, towers)
        disk = towers[source].pop()
        towers[target].append(disk)
        print(f"Move disk {n} from {source} to {target}")
        print_towers(towers)
        hanoi(n - 1, auxiliary, target, source, towers)

def main():
    print("Tower of Hanoi - Recursive with ASCII Visualization")
    n = int(input("Enter number of disks: "))
    towers = {
        'A': list(reversed(range(1, n + 1))),
        'B': [],
        'C': []
    }
    print("Initial State:")
    print_towers(towers)
    hanoi(n, 'A', 'C', 'B', towers)

if __name__ == "__main__":
    main()
```

Key Features:

- Accepts user input for disk count.
- Displays every move with ASCII graphics.
- Fully recursive with readable colored syntax.