ChatGPT

Tower of Hanoi: Recursive Solution in C++

The **Tower of Hanoi** is a classic puzzle involving **three rods** (often named A, B, C) and *n* disks of different sizes [1]. Initially, all disks are stacked on one rod (say A) in decreasing size order (largest on bottom, smallest on top). The goal is to move the entire stack to another rod (say C) **one disk at a time**, following these rules:

- Only one disk may be moved at a time [2].
- Each move takes the **top disk** from one rod and places it on another rod [2].
- No disk may be placed on top of a smaller disk [2].

In other words, at each step you can only move the smallest *current* disk available from one rod to another, and you can never put a larger disk atop a smaller one. For example, with 3 disks the puzzle can always be solved in **7 moves** (which is $2^3 - 1$) [3]. In general, the minimum number of moves required is $2^n - 1$ [3] [4].

Recursive Solution Logic

A recursive solution exploits the self-similar structure of the problem. Denote the rods as **A** (source), **B** (auxiliary), and **C** (destination). To move *n* disks from A to C, the algorithm proceeds in three steps [5]:

1. **Move the top (n-1) disks from rod A to rod B**, using rod C as auxiliary.
2. **Move the largest disk (disk n) from rod A to rod C** (the destination).
3. **Move the (n-1) disks from rod B to rod C**, using rod A as auxiliary.

These steps themselves are done recursively. The *base case* is when *n = 1*: simply move that one disk directly from the source to the destination rod. In pseudo-code: if `n == 1`, do `Move disk 1 from A to C` and return. Otherwise, apply the three steps above recursively. In summary:

- **Base case:** If *n = 1*, move disk 1 from source to destination.
- **Recursive case:** Otherwise, recursively move *n-1* disks, then move disk *n*, then recursively move *n-1* disks again, as outlined above [5].

This pattern ensures all smaller disks are temporarily placed out of the way on the auxiliary rod before moving the largest disk, and then stacked onto the destination rod in correct order. GeeksforGeeks summarizes this pattern as:

> "Shift *n-1* disks from A to B (using C), move the last disk from A to C, then shift *n-1* disks from B to C (using A)" [5].

Using recursion, each call handles one disk and defers the rest to further calls. For example, for *n = 3*, the sequence of calls is: - `towerOfHanoi(3, A, C, B)` calls `towerOfHanoi(2, A, B, C)`,
- which calls `towerOfHanoi(1, A, C, B)` (base case, moves disk 1), then moves disk 2 from A→B, then

calls `towerOfHanoi(1, C, B, A)` (moves disk 1).
- Back in the original call, move disk 3 from A→C, then finally call `towerOfHanoi(2, B, C, A)` to move disks 1–2 from B to C.

This recursive breakdown ensures all rules are obeyed.

C++ Implementation

Below is a commented C++ implementation of the recursive solution:

```cpp
#include <iostream>
using namespace std;

// Recursive function to solve Tower of Hanoi.
// Parameters:
//   n        = number of disks to move
//   from_rod = the rod where disks are initially (source)
//   to_rod   = the rod to move disks to (destination)
//   aux_rod  = the spare rod (auxiliary)
void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod) {
    if (n == 1) {
        // Base case: only one disk to move.
        cout << "Move disk 1 from rod " << from_rod
             << " to rod " << to_rod << endl;
        return;
    }
    // Step 1: move top n-1 disks from 'from_rod' to 'aux_rod'
    towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
    // Step 2: move the remaining largest disk from 'from_rod' to 'to_rod'
    cout << "Move disk " << n << " from rod " << from_rod
         << " to rod " << to_rod << endl;
    // Step 3: move the n-1 disks from 'aux_rod' to 'to_rod'
    towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
}

int main() {
    int N = 3;  // Example: change N for more disks
    // A, B and C are names of the rods
    towerOfHanoi(N, 'A', 'C', 'B');
    return 0;
}
```

Explanation of code: The function `towerOfHanoi(n, A, C, B)` moves `n` disks from rod A to rod C, using B as auxiliary. It first recursively moves `n-1` disks to B, then prints the move of disk *n*, then recursively moves `n-1` disks from B to C. The `main` function sets `N = 3` (for example) and calls the routine.

Example Output (3 Disks)

Running the above code with `N = 3` produces the sequence of moves. For 3 disks, the minimum number of moves is $2^3 - 1 = 7$ [3]. The output (each line is a move) will be:

```
Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 3 from rod A to rod C
Move disk 1 from rod B to rod A
Move disk 2 from rod B to rod C
Move disk 1 from rod A to rod C
```

This sequence matches the expected solution for 3 disks [3]. Each line follows the format *"Move disk k from rod X to rod Y"*, showing how disks are shifted step-by-step under the puzzle rules.

Time and Space Complexity

The recursive Tower of Hanoi has exponential time complexity. The recurrence is **T(n) = 2·T(n–1) + 1**, since each move of *n* disks involves two recursive calls on *n-1* disks plus one move. Solving this (e.g. by back-substitution) gives **$T(n) = 2^n - 1$ moves** [4]. In Big-O terms, the time complexity is **O(2^n)**, which grows exponentially with the number of disks [4]. For example, 5 disks require 31 moves (2^5–1) [4].

The **space complexity** (auxiliary space) is **O(n)**, due to the recursion call stack that can go as deep as *n* calls. At most *n* recursive calls are on the stack at once (each call handling one fewer disk) [6]. Aside from the stack frames, only a constant amount of additional memory is used. GeeksforGeeks summarizes this as *"Auxiliary Space: O(N) (function call stack)"* [6].

Variations and Enhancements

- **Multi-rod variations:** The classic problem uses 3 rods, but one can generalize to more pegs. For example, the 4-peg Tower of Hanoi (called the **Reve's puzzle**) is more complex. An optimal solution for 4 pegs was only proven in 2014, and the general solution uses the *Frame–Stewart algorithm* [7]. In brief, with more rods the strategy is to move some top disks to an intermediate peg (by recursion), then move the remaining disks directly, then move the first group again. The exact optimal move counts become more involved (exponential in a fractional power of *n*).

- **Iterative (non-recursive) solution:** It is possible to solve Tower of Hanoi iteratively. A common approach is to use a loop and an explicit stack or a pattern of moves instead of function calls. For example, one method is to number the moves from 1 to 2^n-1 and at each step move the smallest disk or the only legal move that does not undo the previous one [8]. GeeksforGeeks outlines a cyclic pattern: first compute `totalMoves = 2^n - 1`; if *n* is even, swap the roles of the destination and auxiliary rods, then for each move *i* from 1 to `totalMoves`: 1) if *i%3==1*, move between source and destination,
  2) if *i%3==2*, move between source and auxiliary,

3) if *i%3==0*, move between auxiliary and destination [8].
This deterministic pattern ensures each legal move is made in order. The example animation below illustrates the sequence of moves for 6 disks using one such iterative method:

Figure: Iterative solution move sequence for a 6-disk Tower of Hanoi (source S, auxiliary A, destination D).

- **Visualization/animation:** Many tutorials and applications animate the Tower of Hanoi to help learners visualize the process. You can represent the rods and disks graphically or even textually (e.g. print the state of rods after each move). For instance, some programs use ASCII art or GUI libraries to show disks moving between rods in real time. Animation makes the recursion more intuitive by highlighting how the smaller disks shuttle between rods. (Above, the embedded GIF is an example animation of an iterative move sequence.)

- **Other puzzles and constraints:** Variations include adding rules (like magnetic disks that stick, or limiting the number of moves per day) purely as theoretical or puzzle twists. However, the standard enhancements are the multi-peg generalization and iterative implementations as noted above.

In summary, the Tower of Hanoi puzzle is elegantly solved by recursion in C++. The key idea is to move *n-1* disks out of the way, move the largest disk, and then move the *n-1* disks again. The C++ code is succinct and demonstrates divide-and-conquer. This recursive solution takes $O(2^n)$ time and $O(n)$ space. Iterative algorithms and visual animations are common variations to further explore and demonstrate this classic problem [5] [8] [7].

Sources: The algorithm and analysis above draw on standard references and tutorials [5] [6] [4] [8] [1] [7], which outline the problem description, recursive logic, code examples, complexity, and known variations.

[1] [2] [3] [7] Tower of Hanoi - Wikipedia
https://en.wikipedia.org/wiki/Tower_of_Hanoi

[4] Time Complexity Analysis | Tower Of Hanoi (Recursion) | GeeksforGeeks
https://www.geeksforgeeks.org/time-complexity-analysis-tower-hanoi-recursion/

[5] [6] Program for Tower of Hanoi Algorithm | GeeksforGeeks
https://www.geeksforgeeks.org/c-program-for-tower-of-hanoi/

[8] Iterative Tower of Hanoi | GeeksforGeeks
https://www.geeksforgeeks.org/iterative-tower-of-hanoi/