

The Tower of Hanoi is a classic problem where you have three pegs (A, B, and C) and n disks of increasing size stacked on peg A. The goal is to move the entire stack to peg C, using peg B as an auxiliary peg, following these rules:

1. Only one disk can be moved at a time.
2. A larger disk cannot be placed on top of a smaller disk.
3. All disks must maintain their original order on the destination peg.

Design of the Recursive Solution

The recursive approach breaks the problem into smaller subproblems:

- **Base case:** If there's only one disk ($n=1$), move it directly from the source peg (A) to the destination peg (C).
- **Recursive case:** For n disks:
 1. Move $n-1$ disks from the source peg (A) to the auxiliary peg (B), using the destination peg (C) as a temporary peg.
 2. Move the largest disk (the n th disk) from the source peg (A) to the destination peg (C).
 3. Move the $n-1$ disks from the auxiliary peg (B) to the destination peg (C), using the source peg (A) as a temporary peg.

C Implementation

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

```
// Recursive function to solve Tower of Hanoi
```

```
void towerOfHanoi(int n, char source, char auxiliary, char destination) {
```

```
    // Base case: If only one disk, move it from source to destination
```

```
    if (n == 1) {
```

```
        printf("Move disk 1 from %c to %c\n", source, destination);
```

```
        return;
```

```
    }
```

```
    // Move n-1 disks from source to auxiliary using destination as temp
```

```
    towerOfHanoi(n - 1, source, destination, auxiliary);
```

```
    // Move the nth disk from source to destination
```

```
    printf("Move disk %d from %c to %c\n", n, source, destination);
```

```
    // Move n-1 disks from auxiliary to destination using source as temp
```

```
    towerOfHanoi(n - 1, auxiliary, source, destination);
```

```
}
```

```

int main() {
    int n; // Number of disks
    printf("Enter the number of disks: ");
    scanf("%d", &n);

    if (n <= 0) {
        printf("Number of disks must be positive.\n");
        return 1;
    }

    // Call the recursive function
    printf("Steps to solve Tower of Hanoi with %d disks:\n", n);
    towerOfHanoi(n, 'A', 'B', 'C');

    // Optional: Print the total number of moves (2^n - 1)
    printf("Total moves required: %d\n", (1 << n) - 1);

    return 0;
}

```

General Behavior of the Recursion Stack For a general n , the recursion stack behaves as follows:

- The first recursive call (`towerOfHanoi(n-1, source, destination, auxiliary)`) generates a subtree of calls, reducing n until it reaches 1. This builds the stack up to a depth of n for the first base case (e.g., moving disk 1).
- As base cases are reached, frames are popped, and the function executes the print statement and the second recursive call for each frame.
- The second recursive call creates another subtree, again reducing n to 1.
- The maximum stack depth is $O(n)$, occurring when the recursion reaches the deepest base case (e.g., $n=1$) in the first subtree.

For $n=3$, the stack depth reaches 3 at the first base case, and the total number of stack frames created over the entire execution corresponds to the $2^n - 1 = 7$ moves, but only n frames are active at any one time.