Tower of Hanoi Using Recursion - Quick Revision Notes

Logic of the Given Code:

- The function `hanoi(int n, char a, char b, char c)` solves the Tower of Hanoi problem recursively.
- **Base Condition**:
  - If `n == 0`, the function returns `0`, stopping further recursion.
- **Recursive Calls**:
  - First, move `n-1` disks from source `a` to auxiliary `c` using destination `b`.
  - Print the movement of the `nth` disk from `a` to `b`.
  - Move `n-1` disks from `c` to `b` using `a` as auxiliary.
- This ensures that all disks are moved from `a` to `b` following the rules of the Tower of Hanoi.

Problem Example:

Problem Statement:
Given `n` disks and three rods (A, B, C), move all disks from rod A to rod B using rod C as auxiliary, following these rules:
1. Only one disk can be moved at a time.
2. A larger disk cannot be placed on a smaller disk.

Example (For n = 2):
Steps to move 2 disks from A to B:
1. Move disk 1 from A to C
2. Move disk 2 from A to B
3. Move disk 1 from C to B

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

// Function to solve Tower of Hanoi using recursion
int hanoi(int n, char a, char b, char c) {
    if(n == 0) return 0; // Base case to stop recursion

    hanoi(n-1, a, c, b); // Move n-1 disks from A to C using B
    cout << a << " -> " << b << "\n"; // Move nth disk from A to B
    hanoi(n-1, b, c, a); // Move n-1 disks from C to B using A
}

int main() {
    int n = 2;
    hanoi(n, 'A', 'B', 'C'); // Function call for 2 disks
}
```

Dry Run of the Code (For hanoi(2, 'A', 'B', 'C'))

Function Call	Disk Moved
hanoi(2, A, B, C)	Move 1 from A to C
hanoi(1, A, C, B)	Move 2 from A to B
hanoi(1, C, B, A)	Move 1 from C to B

Final Output for hanoi(2, 'A', 'B', 'C'):

A -> C

A -> B

C -> B

Time Complexity:

- Each move follows the recurrence relation: T(n) = 2T(n-1) + 1
- This results in an exponential complexity of **O(2^n)**.

Key Takeaways:

- The Tower of Hanoi is a classic recursive problem.
- The function moves `n-1` disks to an auxiliary peg before moving the largest disk.
- The minimum number of moves required for `n` disks is **(2^n - 1)**.