1. **Climbing Stairs : https://leetcode.com/problems/climbing-stairs/description/**

```
class Solution(object):
    def climbStairs(self, n):
        memo = {}  # Initialize an empty dictionary to store computed
results
        return self.ways(n, memo)  # Call the ways method with n and
memo as arguments and return the result

    def ways(self, n, memo):
        # Define a helper method named ways which takes n and memo as
input
        if n in memo:  # Check if the result for n is already computed
and stored in memo
            return memo[n]  # Return the stored result directly

        # Base cases: if n is 1 or 2, return 1 or 2 respectively
        if n == 1:
            return 1
        elif n == 2:
            return 2

        # Recursive case: calculate the result for n by summing up
results for n-1 and n-2
        memo[n] = self.ways(n - 1, memo) + self.ways(n - 2, memo)
        return memo[n]  # Store the calculated result in memo and return
it
```

Complexity Analysis:

Time Complexity: O(n), since each value of n is computed only once and stored in the dictionary.

Space Complexity: O(n), due to the usage of dictionary, which stores the results for each value of n.

2. Tower of Hanoi: https://www.geeksforgeeks.org/problems/tower-of-hanoi-1587115621/1

Overview:

*The **tower of Hanoi** is a famous puzzle where we have three rods and N disks. The objective of the puzzle is to move the entire stack to another rod. You are given the number of discs N. Initially, these discs are in the rod 1. You need to print all the steps of discs movement so that all the discs reach the 3rd rod. Also, you need to find the total moves.*

Note: The discs are arranged such that the top disc is numbered 1 and the bottom-most disc is numbered N. Also, all the discs have different sizes and a bigger disc cannot be put on the top of a smaller disc.

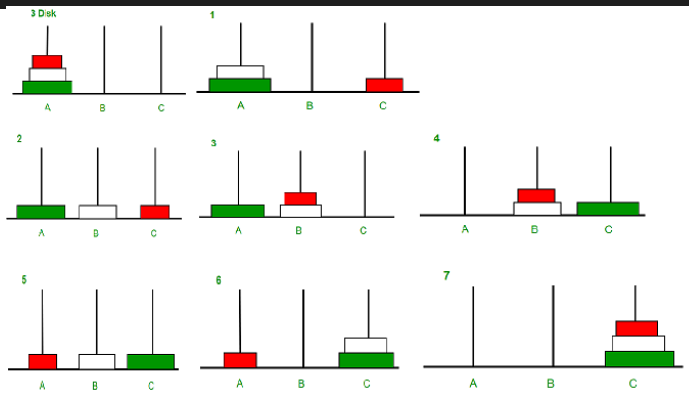
Example 1:

```
Input:
```

```
N = 2
Output:
move disk 1 from rod 1 to rod 2
move disk 2 from rod 1 to rod 3
move disk 1 from rod 2 to rod 3
3
Explanation: For N=2 , steps will be
as follows in the example and total
3 steps will be taken.
```

Example 2:

```
Input:
N = 3
Output:
move disk 1 from rod 1 to rod 3
move disk 2 from rod 1 to rod 2
move disk 1 from rod 3 to rod 2
move disk 3 from rod 1 to rod 3
move disk 1 from rod 2 to rod 1
move disk 2 from rod 2 to rod 3
move disk 1 from rod 1 to rod 3
7
Explanation: For N=3 , steps will be
as follows in the example and total
7 steps will be taken.
```



Constraints:
0 <= N <= 16

Solution:

```
class Solution:
    def toh(self, N, fromm, to, aux):
```

```
        # Base case: If there is only one disk to move, move it
directly from 'fromm' to 'to'
        if N == 1:
            print("move disk " + str(N) + " from rod " + str(fromm)
+ " to rod " + str(to))
            print("\n")
            return 1
             # Return 1 move made

        # Recursive case: Move N-1 disks from 'fromm' to 'aux',
using 'to' as an auxiliary rod
        moves = 0
        moves += self.toh(N - 1, fromm, aux, to)  # Recursive call
        moves += 1  # Increment the total moves count for the
current step
        # Move the remaining largest disk from 'fromm' to 'to'
        print("move disk " + str(N) + " from rod " + str(fromm) + "
to rod " + str(to))

        # Recursive call: Move the N-1 disks from 'aux' to 'to',
using 'fromm' as an auxiliary rod
        moves += self.toh(N - 1, aux, to, fromm)  # Recursive call
        return moves  # Return the total moves made for this step
and all recursive steps

s = Solution()
print(s.toh(3, 1, 3, 2))
```

Complexity Analysis:

Time Complexity: O(2^N), since each call to the toh function generates two recursive calls. The number of function calls made grows exponentially with the number of disks.

Space Complexity: O(n), since the recursion stack can grow up to N levels deep due to the recursive calls, holding information about each move.