Student Name: Aman
UID: 24MCC20015
Branch: MCA(CCD)
Section/Group: 1-A
Semester: 1st
Subject Code: 24CAP-612
Subject Name: DAA LAB

Tower of Hanoi (Puzzle Game)

Aim:- The goal of this project is to develop a Python program that simulates the Tower of Hanoi puzzle. Through this project, you'll understand the recursive approach to problem-solving, learn to implement recursive functions, and gain insight into algorithmic efficiency and complexity.

Task to be done:- This project involves several tasks:

- **Understanding the Puzzle Rules**: Familiarize yourself with the Tower of Hanoi puzzle, its rules, and constraints.
- **Designing the Recursive Solution**: Develop a recursive algorithm that can solve the Tower of Hanoi puzzle for any given number of disks.
- **Implementing the Solution in Python**: Write Python code to execute each move in the sequence generated by the recursive algorithm.
- **Displaying the Moves**: Print each move to show the solution's steps clearly.
- **Visualizing the Solution (Optional)**: Create a graphical representation of the Tower of Hanoi to visualize the movement of disks.

Puzzle Rules :-

The Tower of Hanoi puzzle consists of three rods and a number of disks. The objective is to move all disks from the source rod to the destination rod 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. The disks must be moved from the source rod to the destination rod, using the intermediate rod as a helper.

Algorithm/flowchart:-

The algorithm for solving the Tower of Hanoi puzzle with recursive calls is as follows:

- n: Number of disks
- Source Pole (s_pole)
- Destination Pole (d_pole)
- Intermediate Pole (i_pole)

Steps:

1. Input the number of disks n from the user.
2. Define a recursive function TowerOfHanoi(n, s_pole, d_pole, i_pole):
   - **Base Case**:
     - If n=1, move the disk from s_pole to d_pole and return.
   - **Recursive Case**:
     - Step 1: Call TowerOfHanoi(n-1, s_pole, i_pole, d_pole) to move n−1 disks from the source pole to the intermediate pole.
     - Step 2: Move the n-th (largest) disk from s_pole to d_pole.
     - Step 3: Call TowerOfHanoi(n-1, i_pole, d_pole, s_pole) to move n−1 disks from the intermediate pole to the destination pole.
3. **Execute** the TowerOfHanoi function with the user-defined number of disks and print each move.

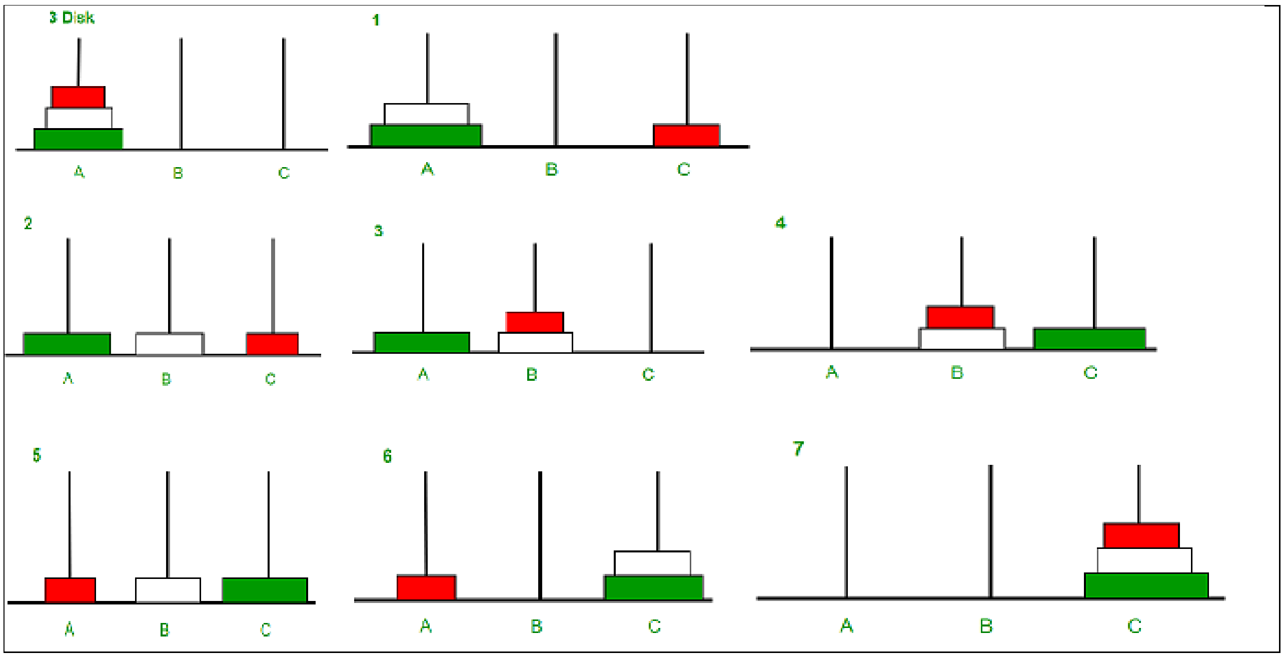
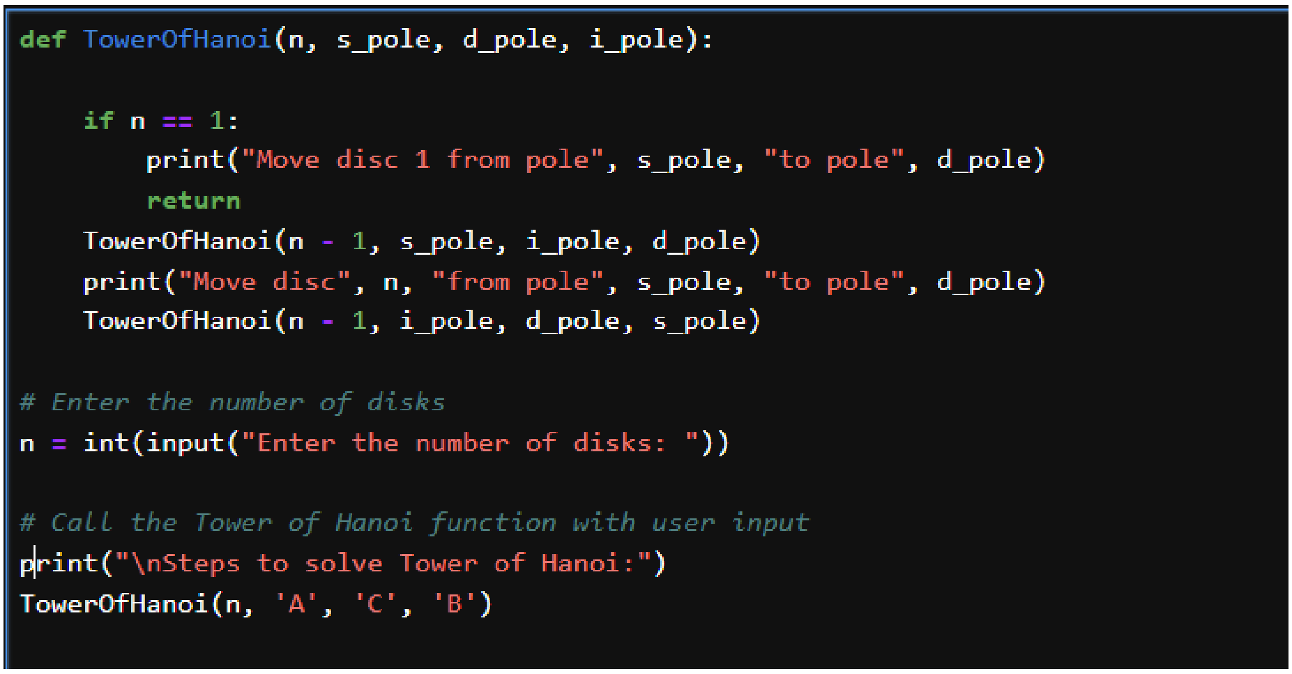


Fig: Tower of Hanoi

Code for experiment:-



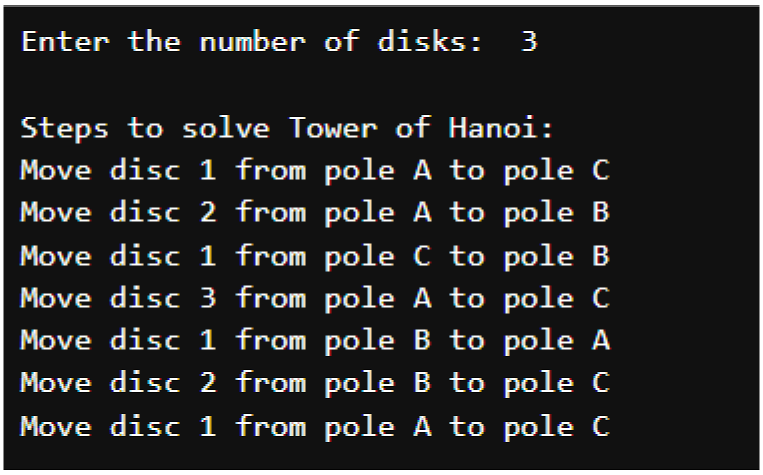
```python
def TowerOfHanoi(n, s_pole, d_pole, i_pole):

    if n == 1:
        print("Move disc 1 from pole", s_pole, "to pole", d_pole)
        return
    TowerOfHanoi(n - 1, s_pole, i_pole, d_pole)
    print("Move disc", n, "from pole", s_pole, "to pole", d_pole)
    TowerOfHanoi(n - 1, i_pole, d_pole, s_pole)

# Enter the number of disks
n = int(input("Enter the number of disks: "))

# Call the Tower of Hanoi function with user input
print("\nSteps to solve Tower of Hanoi:")
TowerOfHanoi(n, 'A', 'C', 'B')
```

Output:



```
Enter the number of disks:  3

Steps to solve Tower of Hanoi:
Move disc 1 from pole A to pole C
Move disc 2 from pole A to pole B
Move disc 1 from pole C to pole B
Move disc 3 from pole A to pole C
Move disc 1 from pole B to pole A
Move disc 2 from pole B to pole C
Move disc 1 from pole A to pole C
```

Learning outcomes:

1. **Understanding Recursive Problem-Solving**: Through implementing the Tower of Hanoi, you gain a deeper understanding of recursion and how complex problems can be broken down into simpler subproblems.
2. **Application of Base and Recursive Cases**: You learn the importance of defining base and recursive cases in a recursive function, which is essential for preventing infinite recursion and for ensuring the function works correctly.
3. **Algorithm Efficiency and Complexity**: The Tower of Hanoi problem demonstrates how recursive algorithms can lead to exponential growth in complexity. This project highlights the O(2n−1) time complexity for recursive solutions, making you more aware of algorithm efficiency.
4. **Time Complexity**: The time complexity is exponential, $O(2^n)$, meaning the algorithm becomes computationally expensive as n increases
5. **Space Complexity**: The space complexity is linear, O(n), because at most n function calls are on the call stack at any given time.
6. **Working with Function Parameters**: By using parameters to represent the source, destination, and intermediate poles, you learn how function parameters can help control and direct the flow of a recursive algorithm.
7. **Problem Solving and Logical Sequencing**: The Tower of Hanoi solution requires moving disks in a specific sequence. This project strengthens logical reasoning and problem-solving skills as you plan and visualize each move in a recursive framework.
8. **User Input and Program Flexibility**: By allowing the user to enter the number of disks, the program is made more flexible and interactive. This practice also enhances skills in handling user input and adjusting program behavior accordingly.