Name: **Dano Alvin John S.** Course & Section: **CPE 311-CPE22S3 (Computational Thinking with Python)** Professor: **NEAL BARTON JAMES MATIRA**

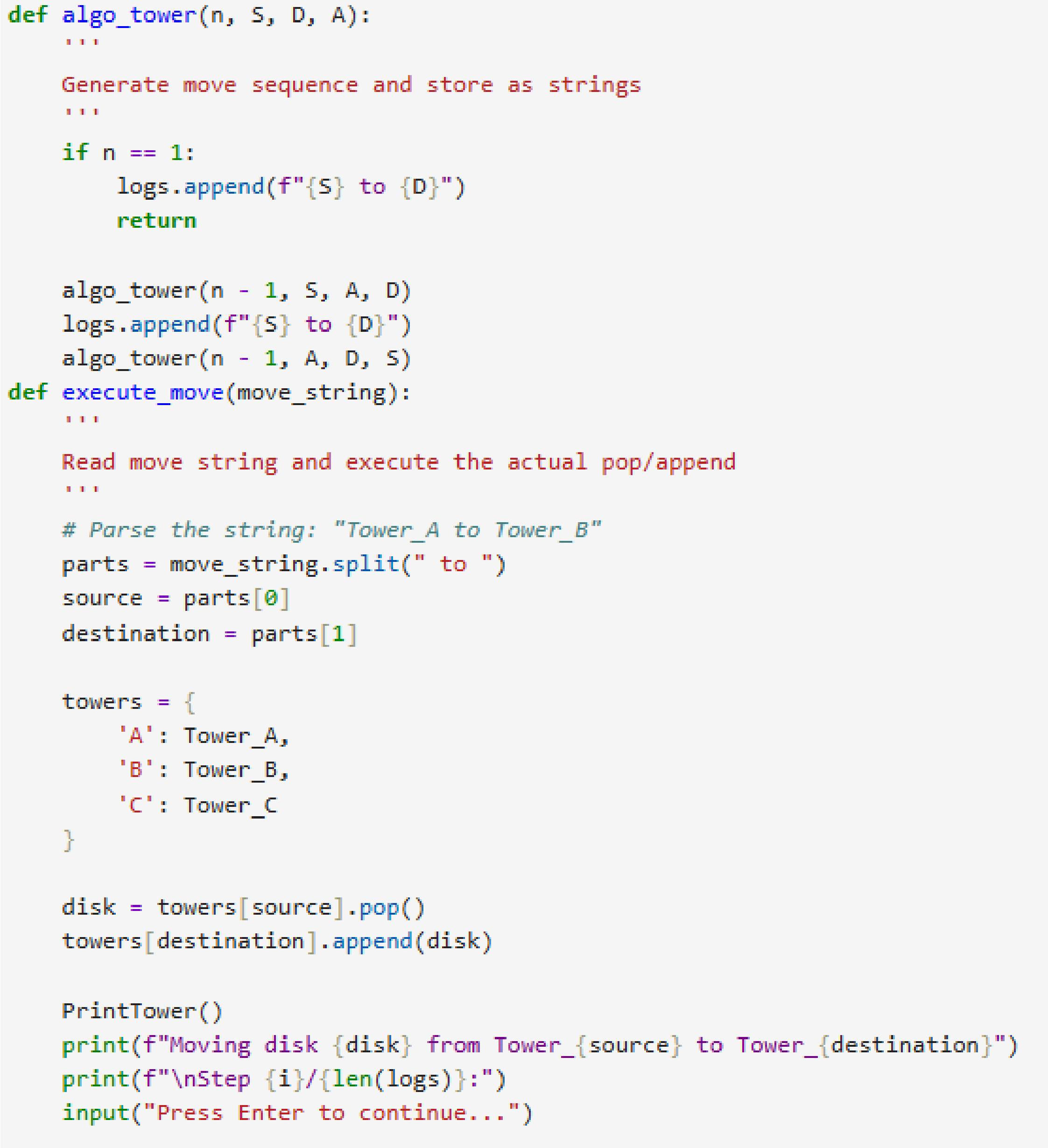
Procedure Implement the algorithm for the Tower of Hanoi Problem as shown above. Write your observations for your written algorithm/solution.

```
import os
def CS():
    os.system('cls' if os.name == 'nt' else 'clear')

# Defining "Towers"
Tower_A = [4, 3, 2, 1]
Tower_B = []
Tower_C = []
logs = []

def PrintTower():
    '''
    Print Tower of Hanoi as UI
    '''
    CS()
    height = 4

    # Print from top to bottom
    for level in range(height - 1, -1, -1):
        row = ""

        # Tower A
        if level < len(Tower_A):
            row += f"  {Tower_A[level]}  "
        else:
            row += "  |  "

        # Tower B
        if level < len(Tower_B):
            row += f"     {Tower_B[level]}  "
        else:
            row += "     |  "

        # Tower C
        if level < len(Tower_C):
            row += f"     {Tower_C[level]}"
        else:
            row += "     |"
        print(row)
    print("—————————————————")
    print("  A       B       C")
    print()
def algo_tower(n, S, D, A):
    '''
    Generate move sequence and store as strings
    '''
    if n == 1:
        logs.append(f"{S} to {D}")
        return

    algo_tower(n - 1, S, A, D)
    logs.append(f"{S} to {D}")
    algo_tower(n - 1, A, D, S)
def execute_move(move_string):
    '''
    Read move string and execute the actual pop/append
    '''
    # Parse the string: "Tower_A to Tower_B"
    parts = move_string.split(" to ")
    source = parts[0]
    destination = parts[1]

    towers = {
        'A': Tower_A,
        'B': Tower_B,
        'C': Tower_C
    }

    disk = towers[source].pop()
    towers[destination].append(disk)

    PrintTower()
    print(f"Moving disk {disk} from Tower_{source} to
Tower_{destination}")
    print(f"\nStep {i}/{len(logs)}:")
    input("Press Enter to continue...")

print("Generating solution for 4 disks...\n")
algo_tower(4, 'A', 'C', 'B')
print(f"Total logs needed: {len(logs)}")

print("Move sequence:", logs)
print("\n" + "="*40 + "\n")
input("Press Enter to start solving...")

print("Initial state:")
PrintTower()
input("Press Enter to start solving...")

for i, move in enumerate(logs, 1):
    execute_move(move)
```

```
Generating solution for 4 disks...

Total logs needed: 15
Move sequence: ['A to B', 'A to C', 'B to C', 'A to B', 'C to A', 'C
to B', 'A to B', 'A to C', 'B to C', 'B to A', 'C to A', 'B to C', 'A
to B', 'A to C', 'B to C']

========================================


Press Enter to start solving...

Initial state:
  1       |       |
  2       |       |
  3       |       |
  4       |       |
─────────────────
  A       B       C


Press Enter to start solving...

  |       |       |
  2       |       |
  3       |       |
  4       1       |
─────────────────
  A       B       C

Moving disk 1 from Tower_A to Tower_B

Step 1/15:

Press Enter to continue...

  |       |       |
  |       |       |
  3       |       |
  4       1       2
─────────────────
  A       B       C

Moving disk 2 from Tower_A to Tower_C

Step 2/15:

Press Enter to continue...

  |       |       |
  |       |       |
  3       |       1
  4       |       2
─────────────────
  A       B       C

Moving disk 1 from Tower_B to Tower_C

Step 3/15:

Press Enter to continue...

  |       |       |
  |       |       |
  |       |       1
  4       3       2
─────────────────
  A       B       C

Moving disk 3 from Tower_A to Tower_B

Step 4/15:

Press Enter to continue...

  |       |       |
  |       |       |
  1       |       |
  4       3       2
─────────────────
  A       B       C

Moving disk 1 from Tower_C to Tower_A

Step 5/15:

Press Enter to continue...

  |       |       |
  |       |       |
  1       2       |
  4       3       |
─────────────────
  A       B       C

Moving disk 2 from Tower_C to Tower_B

Step 6/15:

Press Enter to continue...

  |       |       |
  |       1       |
  |       2       |
  4       3       |
─────────────────
  A       B       C

Moving disk 1 from Tower_A to Tower_B

Step 7/15:

Press Enter to continue...

  |       |       |
  |       1       |
  |       2       |
  |       3       4
─────────────────
  A       B       C

Moving disk 4 from Tower_A to Tower_C

Step 8/15:

Press Enter to continue...

  |       |       |
  |       |       |
  |       2       1
  |       3       4
─────────────────
  A       B       C

Moving disk 1 from Tower_B to Tower_C

Step 9/15:

Press Enter to continue...

  |       |       |
  |       |       |
  |       |       1
  2       3       4
─────────────────
  A       B       C

Moving disk 2 from Tower_B to Tower_A

Step 10/15:

Press Enter to continue...

  |       |       |
  |       |       |
  1       |       |
  2       3       4
─────────────────
  A       B       C

Moving disk 1 from Tower_C to Tower_A

Step 11/15:

Press Enter to continue...

  |       |       |
  |       |       |
  1       |       3
  2       |       4
─────────────────
  A       B       C

Moving disk 3 from Tower_B to Tower_C

Step 12/15:

Press Enter to continue...

  |       |       |
  |       |       |
  |       |       3
  2       1       4
─────────────────
  A       B       C

Moving disk 1 from Tower_A to Tower_B

Step 13/15:

Press Enter to continue...

  |       |       |
  |       |       2
  |       |       3
  |       1       4
─────────────────
  A       B       C

Moving disk 2 from Tower_A to Tower_C

Step 14/15:

Press Enter to continue...

  |       |       1
  |       |       2
  |       |       3
  |       |       4
─────────────────
  A       B       C

Moving disk 1 from Tower_B to Tower_C

Step 15/15:

Press Enter to continue...
```

Observations: First I created the function algo_tower, utilizing recussion to mathematically solve the Tower oh hanoi, implementing divide-and-conquer approach for the Tower of Hanoi problem. I used Bottom-Up on this program and was able to break down and rebuild the problem into more simplified. Time complexity grows exponentially at O(2^n), reflecting the unavoidable cost of the problem. More over we generating moves, visualizing towers, and executing steps makes the design clear and educational. Pre-generating moves before execution highlights the recursive process and allows upfront calculation of total steps ensuring that its mathematically correct before presenting it.

Techniques.



```
def algo_tower(n, S, D, A):
    '''
    Generate move sequence and store as strings
    '''
    if n == 1:
        logs.append(f"{S} to {D}")
        return

    algo_tower(n - 1, S, A, D)
    logs.append(f"{S} to {D}")
    algo_tower(n - 1, A, D, S)
def execute_move(move_string):
    '''
    Read move string and execute the actual pop/append
    '''
    # Parse the string: "Tower_A to Tower_B"
    parts = move_string.split(" to ")
    source = parts[0]
    destination = parts[1]

    towers = {
        'A': Tower_A,
        'B': Tower_B,
        'C': Tower_C
    }

    disk = towers[source].pop()
    towers[destination].append(disk)

    PrintTower()
    print(f"Moving disk {disk} from Tower_{source} to Tower_{destination}")
    print(f"\nStep {i}/{len(logs)}:")
    input("Press Enter to continue...")
```

The function moves n-1 disks to the auxiliary tower, moves the largest disk to the destination, then moves the n-1 disks from the auxiliary to the destination. For 4 disks, this generates exactly 15 moves, which is the optimal minimum number of moves required. The solution achieves the mathematically optimal number of moves, 2^n-1, ensuring correctness.