Assignment On

‘ADVANCED DATA STRUCTURES AND ALGORITHMS’

(Assignment-4)

Submitted by: Suganth C J

Roll No: 2503B09901(MCA).

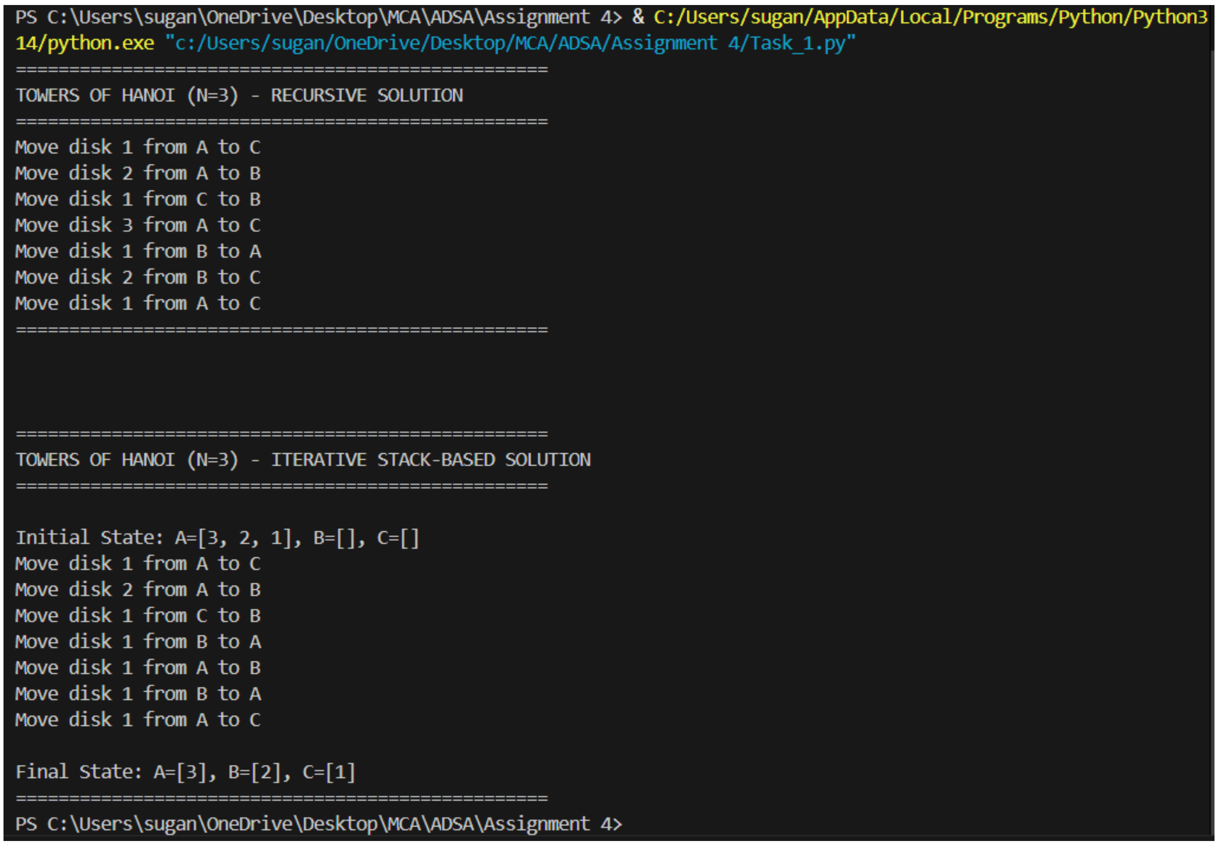
Submitted to: Dr. Rahul Mishra.

Question:
Solve the Towers of Hanoi Problem Recursively. Also, Implement a Stack-Based Iterative Solution.

Code :

```
import sys

# Set a higher recursion limit for safety, although the standard 3 disks won't hit it.
sys.setrecursionlimit(2000)

# --- 1. Recursive Solution ---

def hanoi_recursive(n, source, auxiliary, destination):
    """
    Solves the Towers of Hanoi problem using the classic recursive approach.

    Args:
        n (int): The number of disks to move.
        source (str): The name of the source pole (e.g., 'A').
        auxiliary (str): The name of the auxiliary/temporary pole (e.g., 'B').
        destination (str): The name of the destination pole (e.g., 'C').
    """
    # Base Case: Only one disk left to move
    if n == 1:
        print(f"Move disk 1 from {source} to {destination}")
        return

    # Step 1: Move n-1 disks from Source to Auxiliary, using Destination as auxiliary
    hanoi_recursive(n - 1, source, destination, auxiliary)

    # Step 2: Move the nth disk (the largest remaining) from Source to Destination
    print(f"Move disk {n} from {source} to {destination}")

    # Step 3: Move the n-1 disks from Auxiliary to Destination, using Source as auxiliary
    hanoi_recursive(n - 1, auxiliary, source, destination)

# --- 2. Stack-Based Iterative Solution ---

def _is_legal_move(source_stack, destination_stack):
    """
    Checks if moving the top disk from source to destination is legal.
    A move is legal if the destination stack is empty OR
    the top disk of the source is smaller than the top disk of the destination.
    """
    if not source_stack:
        return False

    source_disk = source_stack[-1]

    if not destination_stack:
        return True

    destination_disk = destination_stack[-1]

    return source_disk < destination_disk

def _transfer_disk(source_stack, destination_stack, source_name, destination_name):
    """
    Attempts to move the top disk between two poles (stacks).
    The move is only executed if it is legal (smaller disk on top of larger).
    """
    if _is_legal_move(source_stack, destination_stack):
        # Perform the move
        disk = source_stack.pop()
        destination_stack.append(disk)
        print(f"Move disk {disk} from {source_name} to {destination_name}")
        return True
    return False

def hanoi_iterative_stack_based(n, source_name='A', auxiliary_name='B', destination_name='C'):
    """
    Solves the Towers of Hanoi iteratively using stacks to represent the poles
    and managing the total number of moves.

    Args:
        n (int): The number of disks to move.
        source_name (str): The name of the source pole.
        auxiliary_name (str): The name of the auxiliary/temporary pole.
        destination_name (str): The name of the destination pole.
    """
    # 1. Initialize the three stacks (Poles)
    # The stacks hold the disk size (1 is smallest, n is largest).
    # Disks are pushed onto A in descending order of size.
    pole_a = list(range(n, 0, -1))
    pole_b = []
    pole_c = []

    poles = {source_name: pole_a, auxiliary_name: pole_b, destination_name: pole_c}

    # 2. Total number of moves is 2^n - 1
    total_moves = (2 ** n) - 1

    # 3. Determine pole order based on N parity
    # If N is even, swap the roles of auxiliary and destination poles
    # for the cyclic movement of the smallest disk.
    if n % 2 == 0:
        # Cyclic order: A -> B -> C -> A
        pole_names = [source_name, auxiliary_name, destination_name]
    else:
        # Cyclic order: A -> C -> B -> A (Standard)
        pole_names = [source_name, destination_name, auxiliary_name]

    print(f"\nInitial State: A={pole_a}, B={pole_b}, C={pole_c}")
    # 4. Loop through the required number of moves
    for i in range(1, total_moves + 1):
        # A. Move the smallest disk (Disk 1) cyclically:
        if i % 2 != 0:
            # The smallest disk always moves every odd step (i=1, 3, 5, ...)

            # Find current and next pole in the cyclic sequence (S -> D -> A or S -> A -> D)
            current_pole_index = (i // 2) % 3
            next_pole_index = (current_pole_index + 1) % 3

            p1_name = pole_names[current_pole_index]
            p2_name = pole_names[next_pole_index]

            p1_stack = poles[p1_name]
            p2_stack = poles[p2_name]

            # Move disk 1 from P1 to P2 if P1 has disk 1, otherwise move from P2 to P1
            # (Ensures the smallest disk is moved cyclically)
            if p1_stack and p1_stack[-1] == 1:
                _transfer_disk(p1_stack, p2_stack, p1_name, p2_name)
            else:
                _transfer_disk(p2_stack, p1_stack, p2_name, p1_name)

        # B. Perform the only legal non-smallest-disk move:
        else:
            # Check all three possible moves between non-smallest poles and execute the first legal one
            # The order of checking (A<->B, B<->C, A<->C) doesn't strictly matter,
            # as only one non-smallest move will be legal at any even step.

            # Check A <-> B
            if _transfer_disk(poles[source_name], poles[auxiliary_name], source_name, auxiliary_name):
                continue
            if _transfer_disk(poles[auxiliary_name], poles[source_name], auxiliary_name, source_name):
                continue

            # Check A <-> C
            if _transfer_disk(poles[source_name], poles[destination_name], source_name,
destination_name):
```

```
            continue
        if _transfer_disk(poles[destination_name], poles[source_name], destination_name, source_name):
            continue

        # Check B <-> C
        if _transfer_disk(poles[auxiliary_name], poles[destination_name], auxiliary_name, destination_name):
            continue
        if _transfer_disk(poles[destination_name], poles[auxiliary_name], destination_name, auxiliary_name):
            continue

    print(f"\nFinal State: A={poles[source_name]}, B={poles[auxiliary_name]}, C={poles[destination_name]}")

# --- Demonstration ---

if __name__ == "__main__":
   NUMBER_OF_DISKS = 3

   # ----------------------------------
   print("=" * 50)
   print(f"TOWERS OF HANOI (N={NUMBER_OF_DISKS}) - RECURSIVE SOLUTION")
   print("=" * 50)
   hanoi_recursive(NUMBER_OF_DISKS, 'A', 'B', 'C')
   print("=" * 50)

   # ----------------------------------
   print("\n" * 2)
   print("=" * 50)
   print(f"TOWERS OF HANOI (N={NUMBER_OF_DISKS}) - ITERATIVE STACK-BASED SOLUTION")
   print("=" * 50)
   hanoi_iterative_stack_based(NUMBER_OF_DISKS, 'A', 'B', 'C')
   print("=" * 50)
```
Output:



```
PS C:\Users\sugan\OneDrive\Desktop\MCA\ADSA\Assignment 4> & C:/Users/sugan/AppData/Local/Programs/Python/Python3
14/python.exe "c:/Users/sugan/OneDrive/Desktop/MCA/ADSA/Assignment 4/Task_1.py"
==================================================
TOWERS OF HANOI (N=3) - RECURSIVE SOLUTION
==================================================
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
==================================================



==================================================
TOWERS OF HANOI (N=3) - ITERATIVE STACK-BASED SOLUTION
==================================================

Initial State: A=[3, 2, 1], B=[], C=[]
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 1 from B to A
Move disk 1 from A to B
Move disk 1 from B to A
Move disk 1 from A to C

Final State: A=[3], B=[2], C=[1]
==================================================
PS C:\Users\sugan\OneDrive\Desktop\MCA\ADSA\Assignment 4>
```