AIML ASSGNMENT # 1:

TOWER OF HANOI

(A)

CODE:

```
def hanoi(n, source, target, auxiliary, moves):
    if n > 0:
        hanoi(n-1, source, auxiliary, target, moves)
        target.append(source.pop())
        moves.append((source.copy(), auxiliary.copy(), target.copy()))
        hanoi(n-1, auxiliary, target, source, moves)
A = [3, 2, 1]  # Initial state for N = 3
B = []
C = []
moves = [(A.copy(), B.copy(), C.copy())]
hanoi(3, A, C, B, moves)
# Print all moves
for move in moves:
    print(f"A: {move[0]}, B: {move[1]}, C: {move[2]}")
print(f"Number of moves: {len(moves)}")
```

```
A: [3, 2], B: [], C: [1]
A: [3], B: [1], C: [2]
A: [], B: [3], C: [2, 1]
A: [], B: [2, 1], C: [3]
A: [2], B: [3], C: [1]
A: [], B: [1], C: [3, 2]
A: [], B: [], C: [3, 2, 1]
Number of moves: 8
PS C:\Users\M.JAHANZAIB\Desktop\AL & ML
```

(B)

AT N=3:

```
def hanoi(n, source, target, auxiliary, moves):
    if n > 0:
        hanoi(n-1, source, auxiliary, target, moves)
        target.append(source.pop())
```

```
        moves.append((source.copy(), auxiliary.copy(), target.copy()))
        hanoi(n-1, auxiliary, target, source, moves)
A = [3, 2, 1]  # Initial state for N = 3
B = []
C = []
moves = [(A.copy(), B.copy(), C.copy())]
hanoi(3, A, C, B, moves)
# Print all moves
for move in moves:
    print(f"A: {move[0]}, B: {move[1]}, C: {move[2]}")
print(f"Number of moves: {len(moves)}")
```

```
A: [], B: [], C: [3, 2, 1]
Number of moves: 8
PS C:\Users\M.JAHANZAIB\Desktop\AL & ML
```

AT N=4:

```
def hanoi(n, source, target, auxiliary, moves):
    if n > 0:
        hanoi(n-1, source, auxiliary, target, moves)
        target.append(source.pop())
        moves.append((source.copy(), auxiliary.copy(), target.copy()))
        hanoi(n-1, auxiliary, target, source, moves)
A = [4,3, 2, 1]  # Initial state for N = 4
B = []
C = []
moves = [(A.copy(), B.copy(), C.copy())]
hanoi(4, A, C, B, moves)
for move in moves:
    print(f"A: {move[0]}, B: {move[1]}, C: {move[2]}")
print(f"Number of moves: {len(moves)}")
```

```
A: [], B: [], C: [4, 3, 2, 1]
Number of moves: 16
PS C:\Users\M.JAHANZAIB\Desktop\AL & ML
```

AT N=5:

```
def hanoi(n, source, target, auxiliary, moves):
    if n > 0:
        hanoi(n-1, source, auxiliary, target, moves)
        target.append(source.pop())
        moves.append((source.copy(), auxiliary.copy(), target.copy()))
        hanoi(n-1, auxiliary, target, source, moves)
A = [5,4,3, 2, 1]  # Initial state for N = 5
B = []
C = []
moves = [(A.copy(), B.copy(), C.copy())]
# Run Towers of Hanoi for N = 5
hanoi(5, A, C, B, moves)
for move in moves:
    print(f"A: {move[0]}, B: {move[1]}, C: {move[2]}")
print(f"Number of moves: {len(moves)}")
```

```
A: [], B: [], C: [5, 4, 3, 2, 1]
Number of moves: 32
PS C:\Users\M.JAHANZAIB\Desktop\AL & ML
```

AT N=6:

```
def hanoi(n, source, target, auxiliary, moves):
    if n > 0:
        hanoi(n-1, source, auxiliary, target, moves)
        target.append(source.pop())
        moves.append((source.copy(), auxiliary.copy(), target.copy()))
        hanoi(n-1, auxiliary, target, source, moves)
A = [6,5,4,3, 2, 1]  # Initial state for N = 6
B = []
```

```
C = []
moves = [(A.copy(), B.copy(), C.copy())]
hanoi(6, A, C, B, moves)
for move in moves:
    print(f"A: {move[0]}, B: {move[1]}, C: {move[2]}")
print(f"Number of moves: {len(moves)}")
```

```
A: [], B: [], C: [6, 5, 4, 3, 2, 1]
Number of moves: 64
PS C:\Users\M.JAHANZAIB\Desktop\AL & ML Github>
```

(C)

For 3 discs, Code utilizes 8 moves, while I utilize 15 moves.

For 4 discs, code utilize just 16 moves, while I took 16 moves.

For 5 discs, code utilize just 32. while I took 86 moves.

For 6 discs, code utilize just 64.

(D)

Time Complexity (O(2^N)):

- The time complexity represents how the algorithm's running time grows as the input size increases.
- In the Towers of Hanoi algorithm, with each additional disc (N), the number of moves required doubles.
- Therefore, the time complexity is exponential and expressed as O(2^N), where N is the number of discs.

Space Complexity (O(N)):

- The space complexity indicates the amount of memory used by the algorithm.
- In the Towers of Hanoi algorithm, the primary factor contributing to space usage is the recursive call stack.
- The maximum depth of the call stack is N (the number of discs), leading to a space complexity of O(N).