## Discussion

The Tower of Hanoi is a mathematical puzzle invented by the French mathematician Edouard LucasLinks to an external site. in 1883.

There are three pegs, source(A), Auxiliary (B) and Destination(C). Peg A contains a set of disks stacked to resemble a tower, with the largest disk at the bottom and the smallest disk at the top. figure 1 Illustrate the initial configuration of the pegs for 3 disks. The objective is to transfer the entire tower of disks in peg A to peg C maintaining the same order of the disks.

Obeying the following rules:

Only one disk can be transfer at a time. Each move consists of taking the upper disk from one of the peg and placing it on the top of another peg i.e. a disk can only be moved if it is the uppermost disk of the peg. Never a larger disk is placed on a smaller disk during the transfer. The solution to the puzzle calls for an application of recursive functions and recurrence relations.

## Procedure

Implement the algorithm for the Tower of Hanoi Problem as shown above.

Write your observations for your written algorithm/solution.

The algorithm first checks if there is only one disk (when n == 1). In this case, it moves the disk directly from the source peg to the destination peg and shows this move. For more than one disk (when n > 1), the algorithm follows a recursive approach. It moves n-1 disks from the source peg to an auxiliary peg, then moves the largest disk from the source peg to the destination peg, and finally moves the n-1 disks from the auxiliary peg to the destination peg recursively. Each move made during the execution is printed to show the sequence of actions taken. The function uses parameters like source, auxiliary, and destination to specify which pegs are involved in moving the disks. This method demonstrates the divide-and-conquer technique, which simplifies the Tower of Hanoi problem by breaking it into smaller parts until reaching a stage that can be directly solved.

```
def tower_of_hanoi(n, source, auxiliary, destination):
    if n == 1:
        print(f"Move disk {n} from {source} to {destination}")
    else:
        tower_of_hanoi(n - 1, source, destination, auxiliary)
        print(f"Move disk {n} from {source} to {destination}")
        tower_of_hanoi(n - 1, auxiliary, source, destination)

n = 3
tower_of_hanoi(n, "Source", "Auxiliary", "Destination")
```

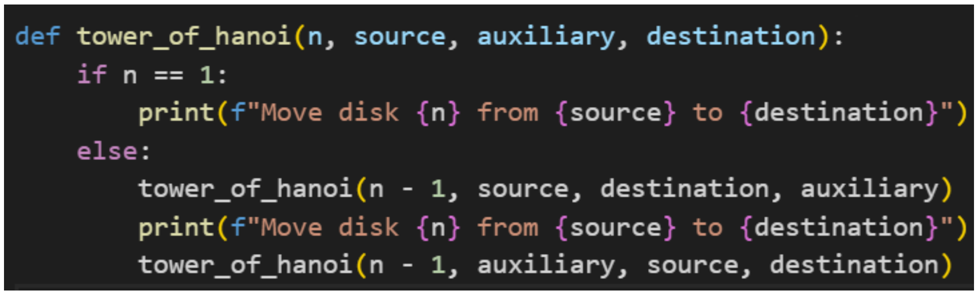
```
Move disk 1 from Source to Destination
Move disk 2 from Source to Auxiliary
Move disk 1 from Destination to Auxiliary
Move disk 3 from Source to Destination
Move disk 1 from Auxiliary to Source
Move disk 2 from Auxiliary to Destination
Move disk 1 from Source to Destination
```

## Supplementary Activity

Explain the programming paradigms/techniques (like recursion or dynamic programming) that you used to solve the given problem.

In this problem I used a method where it repeatedly splits the problem into smaller parts, using recursion. It keeps breaking down the task until it gets to a point where it can directly solve each part. This way of solving problems is useful for tasks that are complicated, as it simplifies them step by step.

Provide screenshots of the techniques and provide a quick analysis.



```python
def tower_of_hanoi(n, source, auxiliary, destination):
    if n == 1:
        print(f"Move disk {n} from {source} to {destination}")
    else:
        tower_of_hanoi(n - 1, source, destination, auxiliary)
        print(f"Move disk {n} from {source} to {destination}")
        tower_of_hanoi(n - 1, auxiliary, source, destination)
```

This algorithm uses a step-by-step approach to move disks from one peg to another. It starts with the smallest step of moving just one disk directly to the destination peg. For more disks, it breaks the task into smaller parts by moving some disks to a temporary peg first. Then it moves the largest disk directly to the destination peg. After that, it moves the remaining disks from the temporary peg to the destination peg. Each move is shown as it happens. This method simplifies the problem by breaking it down into smaller steps until it's easy to solve, demonstrating how this type of problem-solving can handle tasks that involve following a sequence of steps.