**Towers of Hanoi and Recursion: CS14, Prof. Ostheimer**

**Learning objectives:**

- to be introduced to another famous problem in computer science;
- to reinforce the difference between problem specification, algorithm description and algorithm analysis; and to gain further experience in these three phases of development;
- to practice writing recursive Python programs.

**Problem Specification:** We have three pegs and a number of disks of different sizes. The aim is to move from the starting state where all the disks are on the first peg, in size order (smallest at the top) to the goal state where all the disks are on the third peg, also in size order. We are allowed to move one disk at a time, as long as there are no disks on top of it, and as long as we do not move it on the top of a disk that is smaller than it.

- **Input:** $n$, the number of disks for our Towers of Hanoi
- **Output:** a list of moves resulting in a solution to the puzzle

1. Here is one description of a solution to the Towers of Hanoi when $n = 3$.

   ```
   [[1, 3], [1, 2], [3, 2], [1, 3], [2, 1], [2, 3], [1, 3]]
   ```

   Describe the solution in words like "Move blah to blah." and convince yourself it is a valid solution.

2. Figure out a solution when $n = 2$. Write it down using the format above.

3. As we know, a recursive algorithm is one which is based on two ideas:

   - there is a small instance of the puzzle that is easy to solve, and
   - if we can solve a slightly smaller instance of the puzzle, then we can use that solution to solve the larger instance.

   (a) How would you solve the puzzle if $n = 1$, i.e. if there is just one disk?

(b) Suppose your friend knew how to solve the puzzle for $n-1$. Without learning yourself how to solve it for $n-1$ disks, how could you use your friend to solve the puzzle for $n$ disks? Which disks would you ask your friend to move, and to which peg?

4. Here is a Python implementation of the recursive algorithm. Explain how each statement corresponds to your answers to the previous question.

```
def hanoiRecursive(numberOfDisks, fromCol, toCol, usingCol):
   moves = []
   if numberOfDisks == 1:
      moves = moves + [[fromCol, toCol]]
   else:
      moves = hanoiRecursive(numberOfDisks-1, fromCol, usingCol, toCol)
      moves = moves + [[fromCol, toCol]]
      moves = moves + hanoiRecursive(numberOfDisks-1, usingCol, toCol, fromCol)
   return moves
```

5. Trace each of the following calls. When you hit a recursive call, look back at your previous traces to see if you have already traced that call.

```
>>>hanoiRecursive(1,2,3,1)
>>>hanoiRecursive(1,1,2,3)
>>>hanoiRecursive(2,1,3,2)
```

6. Which of the calls above solves the puzzle for $n = 2$?

7. What's all that business about **fromCol**, **toCol** and **usingCol**? Try to understand the order of the parameters for each of the recursive calls.

8. How many moves were generated by the recursive algorithm when $n = 2$? How many when $n = 3$? (Hint: look back to the first question.)

9. Let $H(n)$ be the number of moves that are generated by the program above when there are $n$ disks. What is $H(1)$? $H(2)$? $H(3)$?

10. Find a recursive formula for $H(n)$. To do so you will have to examine the program carefully.

11. Using your recursive formula, make a table of values for $H(n)$ for $n = 1, 2, 3, 4, 5$.

12. Using your table, make a guess at a closed formula for $H(n)$.

13. **At home:** Prove that your closed formula is correct using induction.

14. Find a time complexity function for the program as follows. For each move that is generated, the variable **moves** is updated at most how many times? Assume that each such update takes $10^{-9}$ seconds.

15. Let's say that your computer can run for approximately one week without crashing. How big a Towers of Hanoi problem could you solve in that time? You should be able to estimate and use a standard calculator here, or you can ask me to do some arithmetic using Python.

16. Make a big-O estimate for your time complexity function.

17. True or false: this algorithm has polynomial time complexity.

18. Develop an argument that there is no faster way to solve the Towers of Hanoi than the algorithm presented here.

19. Classify this problem according to the classification scheme we have learned in class.