INTERNATIONAL BACCALAUREATE DIPLOMA PROGRAM

MATHEMATICS ANALYSIS & APPROACHES

INTERNAL ASSESSMENT

Optimization of the moves in the game of Tower of Hanoi

Thesis: Analyzing the relationship between the number of moves to complete the game of Tower of Hanoi and number of disks and rods.

Research Question: What is the general formula for the number of moves to complete Tower of Hanoi with different disk and rod count?

Student code: klx753

Introduction and Rationale:

I always wondered about the shortcuts in games, especially games which let gamer to derive a path, an equation. In our math classroom, I saw a game called Tower of Hanoi which has 3 rods and variable number of disks. While searching the game, I found that there was a legend behind this game. A French mathematician called Edouard Lucas invented this game in the year of 1883. In the game, the player should move all the disks, ordered small to larger, from one rod to another. The only rule is the player cannot move a disk on to a smaller disk. In the beginning of time, there were 64 disks in a Hindu temple which the priests needed to move all disks to another pole. It is believed that the world will vanish as the priests finishes to move all the disks from one pole to another. (Friedman, 2015) Mathematically, the gameplay could be calculated to least amount of moves to reach the goal with an equation.

Aim and Approach:

My aim is to generate a formula which takes the different rod count into account as well as different disk count. In this exploration, I will be using python (high level programming language) to code a solver for the game, so then I can calculate the steps needed in different examples. This will be useful especially in large disk and rod counts.

Methodology:

The task is to generate a formula based on disk and rod count, so I start with 3 rods and different counts of rods to first generate a base path for smallest rod number. Rod count can be at least 3 since one rod possess the disks, second is aimed rod and a third rod for temporary disk holding to arrange.

As an experiment setup, I generated an algorithm which solves for different number of rods and disks because calculating large number of disks on different rod counts is a time taking process which can lead human error. The code is in the appendix.

3 Rods (A, B and C are tags of the rods)

2 Disks = > 3

Move disk 1 from source A to destination C

Move disk 2 from source A to destination B

Move disk 1 from source C to destination B

3 Disks = > 7

Move disk 1 from source A to destination B

Move disk 2 from source A to destination C

Move disk 1 from source B to destination C

Move disk 3 from source A to destination B

Move disk 1 from source C to destination A

Move disk 2 from source C to destination B

Move disk 1 from source A to destination B

4 Disks = > 15

Move disk 1 from source A to destination C
Move disk 2 from source A to destination B
Move disk 1 from source C to destination B
Move disk 3 from source A to destination C
Move disk 1 from source B to destination A
Move disk 2 from source B to destination C
Move disk 1 from source A to destination C
Move disk 4 from source A to destination B
Move disk 1 from source C to destination B
Move disk 2 from source C to destination A
Move disk 1 from source B to destination A
Move disk 3 from source C to destination B
Move disk 1 from source A to destination C
Move disk 2 from source A to destination B
Move disk 1 from source C to destination B

4 Rods (A, B, C and D are tags of the rods)

4 Disks = > 9

Move disk 1 from source A to destination D
Move disk 2 from source A to destination B
Move disk 1 from source D to destination B
Move disk 3 from source A to destination C
Move disk 4 from source A to destination D
Move disk 3 from source C to destination D
Move disk 1 from source B to destination C
Move disk 2 from source B to destination D
Move disk 1 from source C to destination D

5 Disks = > 13

Move disk 1 from source A to destination C
Move disk 2 from source A to destination D
Move disk 3 from source A to destination B
Move disk 2 from source D to destination B
Move disk 1 from source C to destination B
Move disk 4 from source A to destination C
Move disk 5 from source A to destination D
Move disk 4 from source C to destination D
Move disk 1 from source B to destination A
Move disk 2 from source B to destination C
Move disk 3 from source B to destination D
Move disk 2 from source C to destination D
Move disk 1 from source A to destination D

5 Rods (A, B, C, D and E are tags of the rods)

4 Disks = > 7

Move disk 1 from source A to destination B
Move disk 2 from source A to destination D

6 Disks = > 15

Move disk 1 from source A to destination E
Move disk 2 from source A to destination D

Move disk 3 from source A to destination C

Move disk 4 from source A to destination E

Move disk 3 from source C to destination E

Move disk 2 from source D to destination E

Move disk 1 from source B to destination E

Move disk 3 from source A to destination B

Move disk 2 from source D to destination B

Move disk 1 from source E to destination B

Move disk 4 from source A to destination D

Move disk 5 from source A to destination C

Move disk 6 from source A to destination E

Move disk 5 from source C to destination E

Move disk 4 from source D to destination E

Move disk 1 from source B to destination D

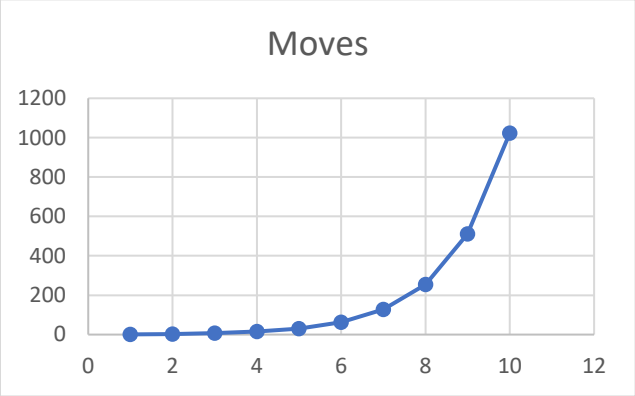
Move disk 2 from source B to destination C

Move disk 3 from source B to destination E

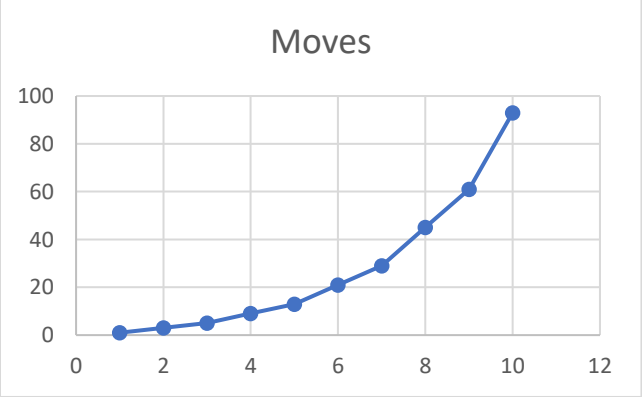
Move disk 2 from source C to destination E

Move disk 1 from source D to destination E

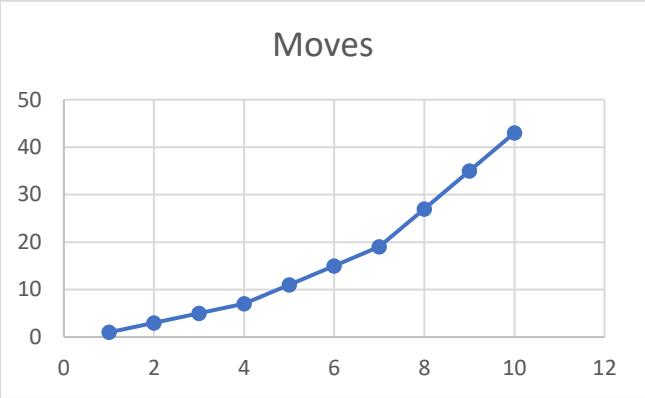
3 Rods



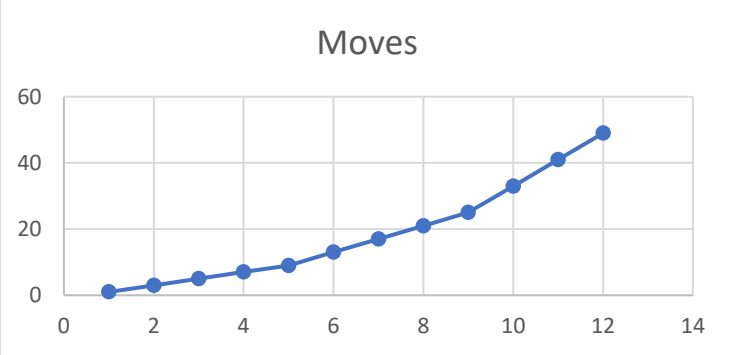
4 Rods



5 Rods



6 Rods



To find equations of different number of rods to derive a general formula. I created a table which shows the direct relationship of move count to disk and rod count.

	3	4	5	6	7
1	1	1	1	1	1
2	3 2^1	3 2^1	3 2^1	3 2^1	3 2^1
3	7 2^2	5 2^1	5 2^1	5 2^1	5 2^1
4	15 2^3	9 2^2	7 2^1	7 2^1	7 2^1
5	31 2^4	13 2^2	11 2^2	9 2^1	9 2^1
6	63 2^5	21 2^3	15 2^2	13 2^2	11 2^1
7	127 2^6	29 2^3	19 2^2	17 2^2	15 2^2
8	255 2^7	45 2^3	27 2^3	21 2^2	19 2^2
9	511 2^8	61 2^4	35 2^3	25 2^2	23 2^2
10	1023 2^9	93 2^4	43 2^3	33 2^3	27 2^2
11	2047 2^{10}	125 2^5	59 2^4	41 2^3	31 2^2
12	4095 2^{11}	189 2^5	75 2^4	49 2^3	39 2^3
13	8191 2^{12}	253 2^6	91 2^4	57 2^3	47 2^3
14	16383 2^{13}	381 2^6	123 2^5	73 2^4	55 2^3
15	32767 2^{14}	509 2^7	155 2^5	89 2^4	63 2^3

We can conclude that there is a relationship between the number of repeating powers of two and rod count. It occurs that powers of two repeats itself by rod count – 2 times. I reason this to be case because of every game has a start and an end rod, so the auxiliary rods are used to make transfers of disks.

From now on n will be the disk count and m will be the rod count.
n = disk count m = rod count

There is an loop of summation calculations. To derive a function which takes all these summations of powers of two we need to relate disk and rod count. Since the game is only playable with 3 or more rod numbers, we can say that there is always a start and an end rod. To calculate, I will take the number of auxiliary rods (rods between start and end rod) which is m – 2. Always 1 disk is needed to play and, in every case, which are meant to finish the game with least number of moves, first disk has one move, so I will take n – 1.

$$\sum_{i=0}^{n-1} 2^{\left\lfloor \left(\frac{i}{m-2}\right) \right\rfloor}$$

This equation takes all values up to disk number -1, and finds corresponding power of two according to rod count, and adds all to give the move count.

However, this summation process is not very person friendly since it still requires deriving the results of the function by the number of disk count. To make a function which directly gives the number of moves as the numbers plugged in, we need to use addition of power of twos.

$2^0 + 2^1 + 2^2 \ldots 2^a = 2^a - 1$

If we calculate the sums of all the powers of twos, and multiply it by the repetition count which is m – 2, we would have a correct result for all the cases where (n - 1) / (m - 2) reminder has no reminder. If we want to get a correct result for all cases, we need to include an additional part.

$$\left(2^{1+\frac{n-1}{m-2}} - 2\right)(m-2)$$

For the part until the last number which (n - 1) / (m - 2) has no reminder.

$$\left(2^{1+\frac{n-1}{m-2}} - 2\right)\left((n-1)\%(m-2)\right)$$

For the part which (n - 1) / (m - 2) give a reminder. We have an additional 1 which is the starting move and always there.

So the general formula would be.

$$\left(2^{1+\frac{n-1}{m-2}} - 2\right)(m-2) + \left(2^{1+\frac{n-1}{m-2}} - 2\right)\left((n-1)\%(m-2)\right) + 1$$

If we expand and simplify.

(-2) (m - 2) + 1 = 5 – 2m

Final formula for calculating the number of moves according to rod and disk count in the game of tower of Hanoi.

$$\left((n-1)\%(m-2) + m - 2\right)2^{1+\frac{n-1}{m-2}} + 5 - 2m$$

Discussion:

I explored the relationship between the quantity of disks and rods and the overall number of movements required to complete the Tower of Hanoi game. Our goal was to develop a formula that would allow us to quickly determine how many moves would be required to win the game.

To do this, we first established a link between the number of disks and rods and the matching power of two by using a summation computation. Next, we used the addition of powers of two to create a formula that was easier to understand.

Our proven algorithm instantaneously predicts the theoretical minimal number of moves required to complete the game, taking into account the number of disks, rods, and auxiliary rods required to complete the game. After several calculations, we arrived to the following formula: (n-1) %(m-2)+m-2) 2(1+(n-1)/(m-2))+5-2m.

This formula offers insight into how the number of disks and rods in play affects the overall number of moves necessary to win the game, which is useful to understand the mathematical concepts in the game. We need to consider that the formula is based on optimum number of moves which means perfect play. Since players can make mistakes, the actual number of moves required may really be higher.

Additionally, we can expand the scope of our research by considering additional game-related factors, such as the disks' sizes and shapes, their initial placement, and the number of competing players, that may have an impact on the overall number of movements required to win. We can also look at the mathematical ideas that underlie the many versions of the game, like the Tower of Brahma and the Tower of London and see how they differ from one another.

Conclusion:

In conclusion, our research into the Tower of Hanoi game has helped us understand the math behind how the game works. We came up with a formula that directly calculates the number of moves needed to finish the game based on the number of disks and rods in the game. This gives us a theoretical basis for understanding the best way to play the game. The number of disks and rods in the game is used to figure out this formula.

But our analysis is limited because it is based on the idea that the gameplay can be made better. More research can be done to find out how things other than the game itself might affect the number of moves needed to win. Even so, our results not only help us understand the Tower of Hanoi game and the math behind it, but they also show how important mathematical thinking and analysis are when trying to solve problems in the real world.

References:

Friedman, E. H. (2015, November 1). The Tower of Hanoi. Scientific American. https://www.scientificamerican.com/article/the-tower-of-hanoi/

Appendices: the code I generated to calculate number of moves to complete the game with different disk and rod count.

```
def th(d,p):
    rods,moves = [str(chr(65+i)) for i in range(p)],[]
    #rod1,rod2 = [i for i in rods],[i for i in rods]
    #rod1[1],rod1[-1],rod2[0],rod2[2] = rod1[-1],rod1[1],rod2[2],rod2[0]
    def tht(n,pegs):
        rod1,rod2 = [i for i in pegs],[i for i in pegs]
        rod1[1],rod1[-1],rod2[0],rod2[2] = rod1[-1],rod1[1],rod2[2],rod2[0]
        if n<1:return
        if n>0 and n<len(pegs)-2:
            order_n = [i for i in range(n-1,0,-1)]+[i for i in range(n)]
            from_n = [0 for i in range(n)]+[-i for i in range(n,1,-1)]
            to_n = [i for i in range(-2,-n-1,-1)]+[-1 for i in range(n)]
            for i in range((n-1)*2+1):
                moves.append([n-order_n[i],pegs[from_n[i]],pegs[to_n[i]]])
        tht(n+2-len(pegs),pegs)
        order_n = [i for i in range(n-1,0,-1)]+[i for i in range(n)]
        from_n = [0 for i in range(n)]+[-i for i in range(n,1,-1)]
        to_n = [i for i in range(-2,-n-1,-1)]+[-1 for i in range(n)]
        print(order_n,from_n,to_n,pegs)
        for i in range((n-1)*2+1):
            moves.append([n-order_n[i],pegs[from_n[i]],pegs[to_n[i]]])
        tht(n+2-len(pegs),pegs[::-1])
    tht(d,rods)
    return moves
```