Algorithms and Datastructures – Tower Of Hanoi

Stefan Ravn van Overeem – stvan13@student.sdu.dk
Martin Staal Steenberg – mstee13@student.sdu.dk

September 24, 2015

1 Implementation

We have implented a solution to the Tower of Hanoi Problem using recursion. The rods have been implemented as a class containing a list of discs on the rod, with functions to place new discs onto the top of the rod, and likewise remove discs from the top. Error checking is done to emit a error message if a large disc is placed on top of a smaller one.

The Tower of Hanoi Game is likewise implemented as a class, containing three Rods. When the rods are constructed, a specified number of discs is placed upon the first rod, while the two others are left empty. A function, called solve is created, which calls the recursive function move_discs, to solve the game.

Algorithm 1 solve function

```
1: function SOLVE
2:     this->MOVE_DISCS(A, B, C, N)
3: end function
```

Algorithm 2 move_discs function

```
1: function MOVE_DISCS(Rod &From, Rod &To, Rod &Last, int N)
2:     if N == 1 then //Base case
3:         Pop top of From, and push it onto To.
4:         return
5:     end if
6:     MOVE_DISCS(From, Last, To, N - 1)
7:     MOVE_DISCS(From, To, Last, 1)
8:     MOVE_DISCS(Last, To, From, N - 1)
9: end function
```

The move_discs function works by first moving the N - 1 top discs from the first rod to the last rod. This means that there now is only one disc left on the first rod, which then easily can be moved to the second and final rod. The discs moved to the last rod is then moved to the final rod, completing the puzzle.

The function calls itself recursively, and reduces the number of discs to move down to one, which is then easily moved to the wanted rod.

2 Time Complexity

To solve the tower N disc have to be moved. For each $N - 1$ disc that have to be moved the prior discs have to be moved and then placed back on top. Which means we get a time complexity of $\mathcal{O}(2^N)$.

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Time [µs]	2	6	0	0	1	1	2	2	3	8	12	23	44	88	175

N	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Time [µs]	346	686	1378	2735	5636	10970	21057	41632	82872	166097	332145	668045	1342962	2612542	5220101

Figure 1:

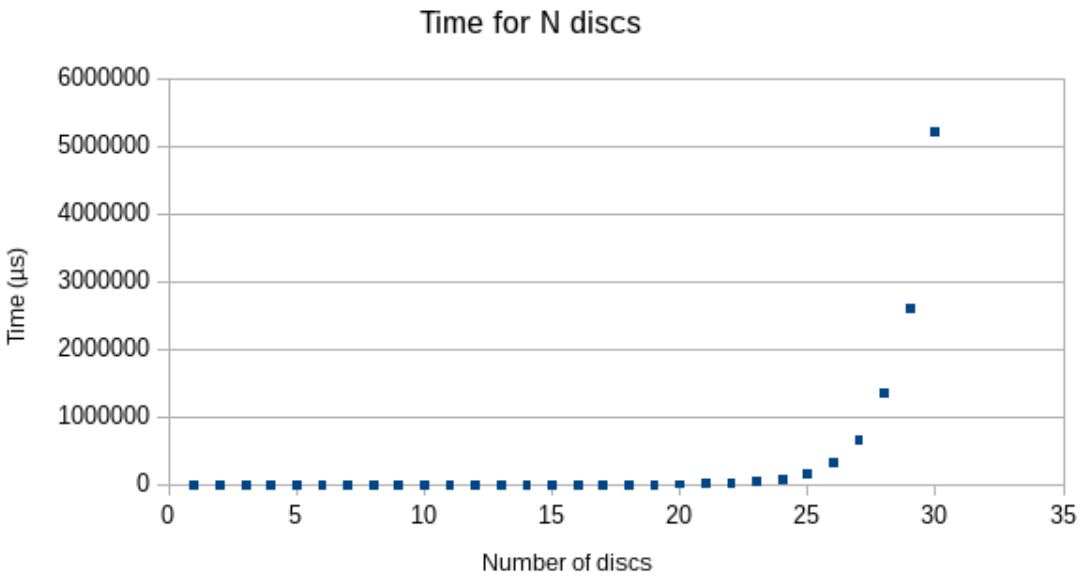

Figure 2:

As can be seen on figure 1 and 2, the time doubles each time N is increased by one. This means that the expected time complexity of $\mathcal{O}(2^N)$, is true for this implementation of the Tower of Hanoi.

The memory complexity of the program is equal to $\mathcal{O}(N)$ because the only thing that have to be stored is the call stack and the rods, which is the size of $2 \cdot N$.

3 Steps

Steps taken by the algorithm with N = 5 can be seen on figure 3

STEP 1: A: 5 4 3 2 B: 1 C:	STEP 2: A: 5 4 3 B: 1 C: 2	STEP 3: A: 5 4 3 B: C: 2 1	STEP 4: A: 5 4 B: 3 C: 2 1
STEP 5: A: 5 4 1 B: 3 C: 2	STEP 6: A: 5 4 1 B: 3 2 C:	STEP 7: A: 5 4 B: 3 2 1 C:	STEP 8: A: 5 B: 3 2 1 C: 4
STEP 9: A: 5 B: 3 2 C: 4 1	STEP 10: A: 5 2 B: 3 C: 4 1	STEP 11: A: 5 2 1 B: 3 C: 4	STEP 12: A: 5 2 1 B: C: 4 3
STEP 13: A: 5 2 B: 1 C: 4 3	STEP 14: A: 5 B: 1 C: 4 3 2	STEP 15: A: 5 B: C: 4 3 2 1	STEP 16: A: B: 5 C: 4 3 2 1
STEP 17: A: 1 B: 5 C: 4 3 2	STEP 18: A: 1 B: 5 2 C: 4 3	STEP 19: A: B: 5 2 1 C: 4 3	STEP 20: A: 3 B: 5 2 1 C: 4
STEP 21: A: 3 B: 5 2 C: 4 1	STEP 22: A: 3 2 B: 5 C: 4 1	STEP 23: A: 3 2 1 B: 5 C: 4	STEP 24: A: 3 2 1 B: 5 4 C:
STEP 25: A: 3 2 B: 5 4 1 C:	STEP 26: A: 3 B: 5 4 1 C: 2	STEP 27: A: 3 B: 5 4 C: 2 1	STEP 28: A: B: 5 4 3 C: 2 1
STEP 29: A: 1 B: 5 4 3 C: 2	STEP 30: A: 1 B: 5 4 3 2 C:	STEP 31: A: B: 5 4 3 2 1 C:	

Figure 3:

4 Code

The code can be found on Github <https://github.com/StefanRvO/ADA/tree/master/Portfolio1>
The code can be compiled using cmake.