The Tower of Hanoi

Emil Göransson

Spring Term 2023

Introduction

This report will cover the implementation of a program that solves the classical mathematical problem Tower of Hanoi though recursion.

Explaining the problem

The way to solve the problem lies in how you define it. Since we are utilizing recursion we first want to define a basecase. In our case the function looks like the following:

The way we write the expression comes from the way we defining the inputs.

```
def hanoi(0, _, _, _) do
  []
end
```

Whenever the number of disks equals 0. We return an empty array. We use this to stop the recursion because anything below 0 disks we aren't interested in.

The second step is to get the function to work for n+1 disks. By studying the problem we can understand that every problem can be broken down into smaller sub-problems that is solved in the same way as every other subproblem. This way we can define the final snippet of code that will solve the Tower of Hanoi problem.

```
    def hanoi(n, from, aux, to) do
  hanoi(n - 1, from, to, aux) ++
    [{:move, from, to}] ++
    hanoi(n - 1, aux, from, to)
end
```

Examples of the code

Lets say we want to solve a tower containing 2 disks. We then call hanoi in the following way:

```
hanoi(4, :a, :b, :c)
```

Which will return the correct set of moves needed to solve the tower of height 4.

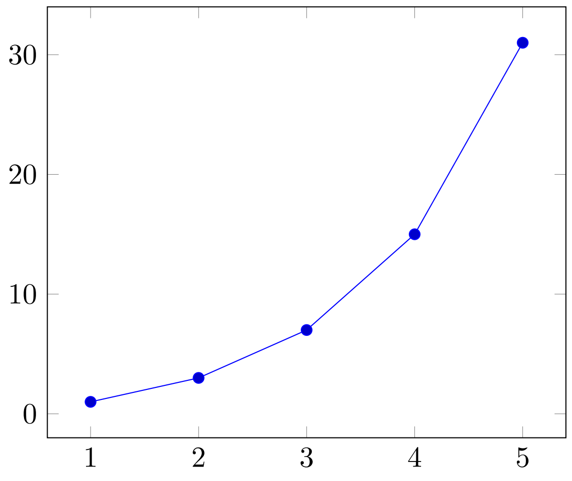
```
        [ {:move, :a, :b}, {:move, :a, :c}, {:move, :b, :c}, {:move, :a, :b},
  {:move, :c, :a}, {:move, :c, :b}, {:move, :a, :b}, {:move, :a, :c},
  {:move, :b, :c}, {:move, :b, :a}, {:move, :c, :a}, {:move, :b, :c},
  {:move, :a, :b}, {:move, :a, :c}, {:move, :b, :c}
]
```

Complexity

By running some tests and counting the amount of moves requiered for different amount of disks the following data is acquired.

n	**#moves**
1	1
2	3
3	7
4	15
5	31

From this data the following graph is plotted.



From the graph we can draw the conclusion that the amount of moves grows at a time complexity of $O(n^2)$.