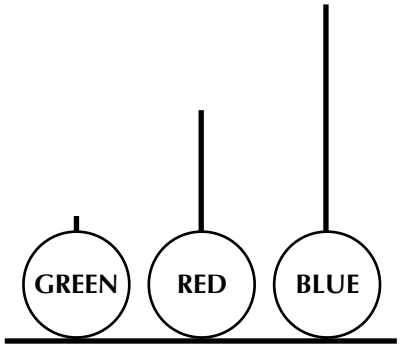
2013 TOWER OF LONDON

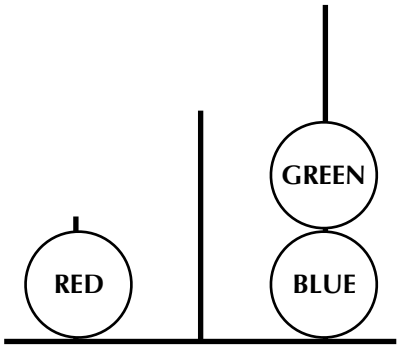
The *Tower of London* test is a puzzle used in neuropsychology for studying problem solving skills. It consists of three pegs, each of which can hold a different number of distinguishable coloured balls. A valid move in the puzzle is to take the top ball from one of the pegs and move it to another peg, if there is space, putting it on top of any balls already on that peg. The objective of the puzzle is to find a sequence of moves to get between two given configurations of balls.

The version of the puzzle most commonly used has three pegs in a row: the left-most able to hold one ball, the middle two balls and the right-most three balls. There are also three balls, which we will denote as red, green and blue.

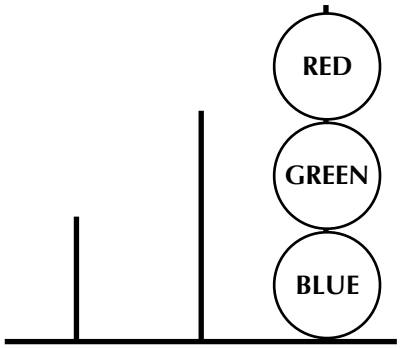
Question 1

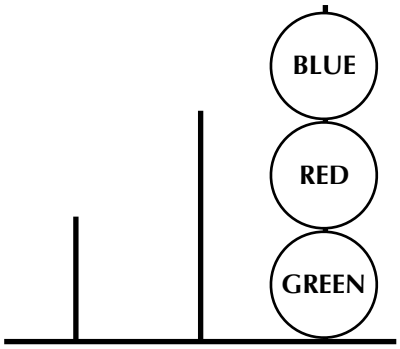
Find a sequence of moves from the position on the left to that on the right:



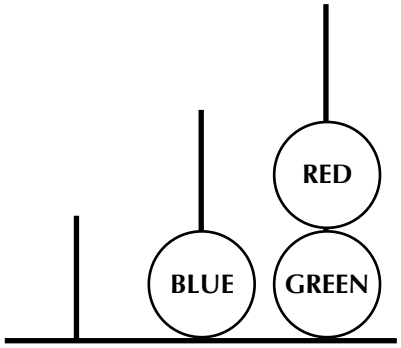


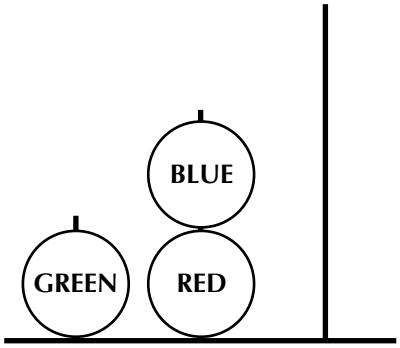
and again for the positions below:





and again for the positions below:





Question 2

Suppose you were setting a generalised version of this problem, as a programming exercise, with p pegs (able to hold *1*, 2, …, p balls respectively), n balls (identified by *1*…*n*) and starting and finishing configurations. Specify a suitable *input* format for the problem and a suitable *output* format.

Question 3

Outline an algorithm for solving the above programming problem.

Question 4

Estimate suitable bounds for n and p which would enable your proposed algorithm to run in a reasonable amount of time. Indicate why these values have been chosen.

Question 5

Is there an approach that you might investigate when looking for an alternative solution, which would run faster or allow larger bounds on n and/or p? If so, *briefly* explain.

Question 6

Give some examples of how the test might be changed to give a variant of the problem, still suitable as a programming exercise.

Question 1 Redux

Find a *shortest* sequence of moves from the position on the left to that on the right and indicate how many other solutions exist with this number of moves. You do *not* need to solve this problem using your algorithm(s), although you are free to do this if you wish.