Math 7h	Professor: Padraic Bartlett

Homework 2: Sorting Algorithms

Due 10/15/13, at the start of class — *UCSB 2013*

Instructions: Choose **one** of the problems below, and work on it until either:

1. You solve the problem, or
2. You have spent about 90 minutes working seriously on the problem.

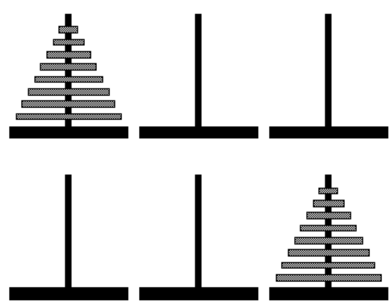
Homework Problems

1. Create an algorithm that takes as input any configuration of chess pieces on a chessboard along with a player's turn, and outputs which player will win if both play perfectly. (Hint: this does not need to be a particuarly fast algorithm. In fact, it probably needs to be insanely slow.)

2. The Towers of Hanoi is the following puzzle: Start with 3 rods. On one rod, place n disks with radii $1, 2, \ldots n$, so that the disk with radius n is on the bottom, the disk with radius $n-1$ is on top of that disk, and so on/so forth.

   The goal of this puzzle is to move all of the disks from one rod to another rod, obeying the following rules:

   - You can move only one disk at a time.
   - Each move consists of taking the top disk off of some rod and placing it on another rod.
   - You cannot place a disk A on top of any disk B with radius smaller than A.

   Find a recursive algorithm for solving this puzzle! How long does it take to complete your solution? Suppose that you can perform a move once every second, and you can perform moves until the heat death of the universe (10^{100} years, say.) What is the largest puzzle you can solve?

3. Consider the following algorithm (Stoogesort[1]!) for sorting a list: Take as input a list $L = (l_1, \ldots l_n)$.

   - If your list contains one or two elements, sort it by just looking at the list.
   - Otherwise, the list contains ≥ 3 elements. Let $M = \lceil 2/3 \rceil$.
   - Stoogesort the list $(l_1, \ldots l_m$.
   - Stoogesort the list $(l_{n-m}, \ldots l_n)$.
   - Stoogesort the list $(l_1, \ldots l_m$.

   Prove that this algorithm sorts any list.

4. Suppose you have a stack of n pancakes of different sizes. You want to sort these pancakes so that smaller pancakes are on top of larger pancakes!

   However, the only tool you have to do this is a spatula. The spatula can **flip** pancakes, as described here:

[1] Named after the comedy routines of the Three Stooges; specifically, the ones where each stooge hits the other two.

- Suppose we have a stack of pancakes. Write this stack as an ordered list $(p_1, \ldots p_n)$, where the first element in our list is at the bottom, the second is directly on top of the first, and so on/so forth.
- We can insert our spatula at any point in the stack. From there, we can **flip** all of the entries above where we put our spatula. For example, suppose we have the stack $(p_1, p_2, p_3, p_4, p_5, p_6)$. We could insert our spatula between pancakes p_3 and p_4, and flip the stack (p_4, p_5, p_6) to get the new arrangement

$$(p_1, p_2, p_3, p_6, p_5, p_4).$$

(Fun fact: the one and only research paper written by Bill Gates studied this problem.) Describe an algorithm to sort an arbitrary stack of n pancakes, using as few flips as possible. How many flips does your algorithm need, in the worst-case scenario?

5. Consider the following recursive algorithm $Factor(n)$ for finding $n!$, given a nonnegative integer input n:

   - If $n = 0$ or 1, return 1.
   - Otherwise, return $n \cdot Factor(n-1)$.

   (a) How many steps does this take to run?

   (b) How many bits are required to write $n!$ in binary, roughly speaking? (Use Stirling's approximation, which says that $n! \approx \sqrt{2\pi n} \cdot (n/e)^n$.)

   (c) Given your answer above, you might believe that the run time you calculated in (a) is far too large for such a simple task! Consider instead the following better algorithm $Factor2(n, m)$ which computes $(n!)/(n-m)!$:

   - If $m = 0$, return 1.
   - Othewise, if $m = 1$, return n.
   - Otherwise, return $Factor2(n, \lfloor m/2 \rfloor) \cdot Factor2(n - \lfloor m/2 \rfloor, \lceil m/2 \rceil)$.

   How many steps does $Factor2(n, n)$ take to calculate $n!/0!$, roughly?