(Some simple variations on the Towers of Hanoi problem)

Recall that in class we wrote recursive algorithms in C for two Towers of Hanoi problems, namely the standard classical problem and the variation we called "Linear Hanoi" (so named because the three towers were considered as arranged in a row and rings could not be moved directly from one end tower to the other). For definitiveness, and for reasons that will become clear, let's suppose we have implemented our algorithms in such a manner that towers are of type **int**, that is, we suppose our two function headers are:

```
void hanoi ( int n, int from, int to, int spare )
```

and

```
void linear_hanoi ( int n, int from, int to, int spare )
```

for the Hanoi and Linear Hanoi problems respectively.

The following problems "build on" the code we've already written, and on each other. Feel free, therefore, to call upon hanoi, linear-hanoi, and earlier problem solutions to solve later ones!

(1). (a). Write a recursive algorithm

```
void mid_to_end ( int n, int from, int to, int spare )
```

to do the "second half" of the Linear Hanoi problem, i.e., move n rings from the middle tower (*from*) to one of the end towers (*to*) using the other end tower (*spare*) as a spare, following the usual rules of Linear Hanoi.

(b). Write a tail-recursive or iterative algorithm

```
void end_to_mid ( int n, int from, int to, int spare )
```

to do the "first half" of the Linear Hanoi problem, i.e., move n rings from one of the end towers (*from*) to the middle tower (*to*) using the other end tower (*spare*) as a spare.

(2). (A Reverse Hanoi problem): Suppose we are given n rings stacked in usual Hanoi fashion on one of three towers (*from*). Suppose also that we are given an array (indexed from 1 through n) of destinations, each element of which = 1, 2, or 3, and which indicates which tower each ring is to end up on. (Thus, for example, if destination[5] =3, we would want ring 5 to end up on tower #3. For simplicity, we assume the three towers are

(continued)

numbered 1,2,3.) We wish to move the rings one at a time, following the usual Hanoi rule of "no large on small", so that each ring ends up on its correct destination tower.

(a). Write a tail-recursive or iterative algorithm

```
void reverse_hanoi ( int n, int from )
```

to solve this problem. (Hints: The destination array should be thought of as global, i.e., there's no need to put it on the parameter list. Also, a very nice convenience resulting from numbering our towers 1,2,3 is that, if variables x and y are two of the three towers, then the third tower is numbered 6-x-y.)

(b). At worst, how many moves does reverse_hanoi take to solve an n ring problem? **Prove your conjecture.**

(3). (A Reverse Linear Hanoi problem): This problem closely resembles the previous one. Once again, suppose we are given n rings stacked in usual Hanoi fashion on one of the three towers (*from*). Suppose also that we are given an array (indexed from 1 through n) of destinations, each element of which = 1, 2, or 3, and which tells us which tower each ring is to end up on. (Thus, for example, if destination[6]=2, we would want ring 6 to end up on tower #2. For simplicity, we assume the three towers are numbered 1,2,3, with tower 2 being the **middle** tower.) We wish to move the rings one at a time, following the usual **Linear** Hanoi rules of "no large on small" and "no jumping over the middle tower", so that each ring ends up on its correct destination tower.

Write a tail-recursive or iterative algorithm

```
void reverse_linear_hanoi ( int n, int from )
```

to solve this problem.

[**Note**: You can, of course, test all your algorithms by actually implementing them on a computer! However, you need not do so if you are secure in your logic and in your C coding skills, i.e., if you feel confident enough that you do not need a **machine** to validate or invalidate your thought processes!