1) (10 pts) DSN (Recursive Coding)

The Towers of Hanoi Problem involves starting with a tower of ***n*** disks (of all different radii) on a single pole and transferring those disks to a different pole. There are only 3 poles that can hold the disks. The disks can only be moved one at a time, from one pole to another, and a disk must always be the only disk on the pole or be placed on top of a disk that is larger. The initial configuration of disks starts with all of the disks sorted in order on a single pole, with the smallest disk on top. For the purposes of this problem, let the radii of the disks be 1 cm, 2 cm, 3 cm, …, ***n*** cm. Let the cost of a single move simply equal the radius of the disk being moved. Define the cost of a solution to the puzzle to be the sum of the costs of the moves to solve the puzzle. Complete the function below, ***using recursion***, so that it takes in a single integer, ***n,*** the number of disks for the puzzle, and returns the minimal cost for transferring the ***n*** disks from the starting pole to either of the other poles. (Note: the actual value of the starting and ending poles don't affect the answer, so long as they are different poles. The code is relatively short and one can do some math to get an O(1) solution, but what is being tested here is the ability to take a problem, break it down recursively, and implement that solution. Thus, the grading criteria described below is focused on rewarding the skill that's being tested, though an alternate, more efficient solution exists.)

Grading Note: Any iterative or non-recursive solution will get a maximum of 3 points, a recursive solution that takes θ(2^n) time will get a maximum of 7 points. To receive full credit, your solution must be recursive and run in O(n) time, where n represents the input value to the function.

```
int towersCost(int n) {

    // Grading: 3 pts, 1 pt if, 1 pt 1, 1 pt ret could also be
    // if (n == 0) return 0;
    if (n == 1) return 1;

    // Grading: 1 pt ret, 2 pts 2*, 2 pts rec call, 1 pt +, 1 pt n
    return 2*towersCost(n-1) + n;

}
```