COP 3502 Suggested Program Edits: Recursion (Week 6 Programs)

1) Look up the Lucas Numbers and write a recursive function that takes in a single non-negative integer, n, and returns the n^{th} Lucas Number.

2) Create a struct of your own and a compareTo function for that struct that takes in two pointers to the struct and returns a negative integer if the first struct "comes before" the second, returns a positive integer if the first struct "comes after" the second, and 0 if the two structs are equal. Then, write a recursive binary search which determines if a desired struct is in a sorted array of the structs.

3) Edit the Water program so that not only does it count the bodies of water, but it also calculates the number of squares in each body of water and prints these out as each individual floodfill is done. (Thus, these prints will occur before the print of the number of regions.)

4) Challenge Problem: Edit the permutation algorithm so that if the objects that are being permutated are repeated, multiple permutations with the repeated objects swapped are not printed. For example, edit the algorithm so that when printing all permutations of "MEME", only six things print: "EEMM", "EMEM", "EMME", "MEEM", "MEME" and "MMEE", instead of 24 (four copies of each of these).

5) Challenge Problem: Consider solving the Towers of Hanoi with four poles instead of three, for n disks. Instead of printing out the moves, just try to calculate a valid path with as few moves as possible. One way to do this (this may not minimize the answer but it proves that the actual minimum is what this algorithm finds or lower) is for n disks, to recursively try solving the puzzle for four poles on k disks, placing the k disks all on an intermediate stack (there are two options). Then, move the remaining n-k disks to the final stack, which takes exactly $2^{n-k} - 1$ moves, since we can't use the fourth pole any more. Then, again, recursively move the k disks from the intermediate tower to the final tower. Note that the recursion here is always solving the puzzle with four poles. The problem is, we don't know what value of k is the best!!! So, in the recursion, try all values of k, from 1 to n-1, and pick the one that minimizes the solution and return it. (Note: This techniques runs much faster with something called memoization which we teach in CS2, so for the time being, you will probably only be able to test this code on an input size of about n = 12 or so, maybe a couple more than that at most.)