COP 3502 Suggested Program Edits: Recursion (Week 5 Programs)

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

2) Write a recursive multiplication function that works similarly to the fast modular exponentiation program. Test your function appropriately. Your function should mostly add and recurse.

3) 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.

4) Challenge Problem: Write a recursive function that takes in two strings: search and target. The function should count how many times target appears as a subsequence in search. For example, if search = "mississippi" and target = "is", then the function should return 6 because there are 6 subsequences of "is" in "Mississippi". A subsequence is a subset of the characters of a string in the same order they appear. Here are each of the subsequences "is" highlighted: "mississippi", "mississippi", "mississippi", "mississippi", "mississippi", and "mississippi". Here is the function prototype:

```
int numSubsequences(char* search, char* target);
```

Note that using pointer arithmetic is required for this question. (Note: search+1 is the substring of the string pointed to by search all except for the first letter.)

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.)