Problem 1: Fibonacci

Iteration Vs. Recursion

After testing both programs, fib.iteration.c & fib.recursion.c, it is clear that fib.iteration.c is faster. When computing large Fibonacci terms, recursion is extremely slow, and iteration is extremely fast. Hence, iteration is more suitable for this problem. It is better to use iteration to compute Fibonacci numbers. For more information, refer to the table below.

	Execution Time In Seconds (Time It Takes To Calc. nth Term)	
n (Fibonacci Term)	Recursion	Iteration
20	0.000322	0.000020
25	0.002747	0.000022
30	0.017123	0.000026
35	0.177530	0.000030
40	1.907370	0.000033
45	21.170792	0.000035

If you were to plot the data from this table, you would get an exponential curve for recursion, and a linear graph for iteration. Computations should take constant time, or linear time. They should never take exponential time. This is why iteration is better than recursion for calculating Fibonacci numbers.

In terms of size of code (i.e. How many lines), both programs are relatively small and near the same length. Iteration takes 30 lines, and Recursion takes 34.

Recursion is slower than iteration (in this case), because stores all function calls in the stack. Because of this, a lot of memory is used and constantly written; especially for large numbers. Iteration, on the other hand, requires 4 variables and only manipulates those. Hence it's much quicker

References:

- https://www.programiz.com/c-programming/examples/fibonacci-series
- https://www.tutorialspoint.com/data_structures_algorithms/fibonacci_recursive_program_in_c.htm
- https://www.advanced-ict.info/programming/recursion.html
- https://stackoverflow.com/questions/5248915/execution-time-of-c-program

Problem 2: Tower Of Hanoi

After testing the programs, it is clear that the recursive solution for the tower of Hanoi is much better than the iterative solution. Both solutions require the same number of executions/calls, 2^n – 1, where n is the number of disks. In this case, there are many benefits to using recursion over iteration, such as:

1. Faster execution time
    a. The recursive function is much faster than its iterative counterpart. Even though both require 2^n – 1 executions, recursion is faster due to simplicity. Plus, the recursive solution uses less memory compared to the iterative solution. The iterative solution has tons of variables to keep track of and manipulate. This is taxing on the CPU and RAM.
2. Significantly less lines of code
    a. The recursive function is 8 lines long, while the iterative function is over 26 lines long. In this case, less is more! The benefits of less lines are tremendous!
3. Easier to maintain
    a. Since the recursive code is a few lines long, it's easier to maintain than its iterative counterpart that is dozens of lines long. Also, the recursive code is easier to troubleshoot and debug in case something goes wrong.
4. Easier to write
    a. The iterative code requires using complex concepts like multi-dimensional arrays, stack data structures, and using malloc. This makes it significantly harder to write, while the recursive implementation is just a few lines long.

Recursion is good for things like divide and conquer. The tower of Hanoi is an excellent example. This assignment proves that iteration isn't always the best choice and sometimes recursion is superior.

My iterative code for this problem isn't as robust as the recursion code. This is because the size of the arrays need to be changed when dealing with a different number of disks. The size of the array needs to be changed and all subsequent things that rely on it. This is a huge hassle, especially when it comes to code maintainability and longevity.

Creating the iterative solution for Tower Of Hanoi was extremely difficult. In fact, this is the hardest CS problem I've done in my life. I ended up with two solutions for this. My first solution (6.hanoi.iteration.bad.c), involved creating two arrays that contain numbers. These numbers correspond to when the disks are shifted from right to left, instead of the normal left to right. This is a bad idea because it doesn't work for 8 and more disks. My second solution (7.hanoi.iteration.good.c) involved using arrays to simulate the towers. This required a lot of code and problem solving. The iterative loop is probably 150 lines long, and it doesn't even include the helper functions. If you tried to do this using the stack, you'd end up with a lengthy solution. Recursion is the superior solution to this problem, by miles! Nothing can beat it!

References:

- https://www.geeksforgeeks.org/c-program-for-tower-of-hanoi/
- https://www.quora.com/Can-a-Tower-of-Hanoi-program-in-C-be-used-by-loops-instead-of-recursion-of-functions-If-yes-then-how
- https://www.youtube.com/watch?v=rVPuzFYlfYE
- https://www.tutorialspoint.com/cprogramming/c_multi_dimensional_arrays.htm
- https://www.quora.com/Can-a-Tower-of-Hanoi-program-in-C-be-used-by-loops-instead-of-recursion-of-functions-If-yes-then-how
- https://en.wikipedia.org/wiki/Tower_of_Hanoi#Iterative_solution
- https://0code.blogspot.com/2012/08/Tower-Of-Hanoi-Iterative.html
- https://codereview.stackexchange.com/questions/137997/tower-of-hanoi-without-recursion
- https://stackoverflow.com/questions/15849656/hanoi-towers-iterative-using-lists
- https://www.quora.com/Can-I-solve-The-Tower-of-Hanoi-problem-using-iteration-instead-of-recursion
- https://stackoverflow.com/questions/25548735/implementation-of-tower-of-hanoi-iterative-procedure
- https://waset.org/publications/11095/on-the-solution-of-the-towers-of-hanoi-problem
- https://www.youtube.com/watch?v=2SUvWfNJSsM

Note: I used CPU time instead of clock time because CPU time (in this case) is more reliable, as we are only interested in how long each block of code takes. Clock time accounts for time wasted by the user, which is something we don't want. Also, CPU time doesn't account for time wasted by other processes or system calls. Our goal is to measure the time it takes for the block code.