A really simple project:

Four methods, of two operations!

First operation:

Code recursively and iteratively the Fibonacci sequence to 30 values, capturing the number of "executions" or "steps" (the "plusses") so you can see how effective iteration and recursion are for this formula.

Second operation:

Code recursively and iteratively the mathematical exponentiation function, given a base and an exponent. The base can be a double, but the exponent must be an integer, but it CAN be negative. Keep track of the "execution" steps in the form of the "multiplications" here. Recognize, in this case, you can do a cool recursive trick when the exponent is even of returning the same function on base doubled but the exponent halved.... this should be in the textbook, and we'll go over it when we're back from break. You can get started on the other three of the four well before this one, though! And, of course, pay attention to the weird cases (base of zero, exponent of zero, etc).

# What I learned

Fibonacci is hard to spell.

I used Wikipedia implementations directly to save time, building my routines using their pseudocode. (But did not copy past any actual code!)

I also did not focus much on the design of them, hence the manual resetting. I have a version with a function pointer constructor in the works, but wanted to focus on getting the assignment done and not the SWE of the design.

The interesting thing here came from the algorithms. Now, it was obvious after our discussions in class that the recursive algorithm for Fibonacci would be less efficient without optimization, but the difference was a lot greater than I was expecting. In retrospect, logs grow rapidly, so it would be expected.

The difference was a lot harder to see in the squaring, but this was due to the nature of numbers. They are bounded by the machine and grow rapidly, so I was actually running into representation issues before I saw too great a difference in the performance, but the difference was there.

In both cases the recursive mechanism introduce unneeded overhead, and prove the point that tail recursion is wasteful very well.