Mod 5: Fibonacci

# Project

Our project for Module 5 is to implement a Fibonacci method - comparing both recursive and iterative versions.

# Tests

For this project I wrote my Fibonacci methods in Java. In the program I ran both recursive and iterative methods and used *System.nanoTime()* to record their run times in nanoseconds.

An issue I ran into while running tests was that the first number in my recursive method was excessively high. It’s expected that your runtime will increase with the size of the input, but the first and smallest input shouldn’t be the highest. After some digging around, I found out this is an issue with Java, specifically JIT (Just in Time), compiling of your code. The simple workaround is to run your method once, without saving the data, and then run the other methods recording as usual. The first run has the risk of having bad data, an outlier, but once it runs the other methods should perform as expected.

# Code

# Outcome

The Fibonacci sequence is the sum of the two numbers before it. When given an input, our methods will calculate the Fibonacci number.

Recursion handles this by calling itself, which for smaller numbers isn’t a problem. But that changes quickly when you recalculate all the Fibonacci numbers that we already have solved as that input number gets larger. This greatly increases your processing time makes this an inefficient way to solve your Fibonacci Sequence.

A better approach is to store the calculated values. This is where the iterative method comes in. We only need to calculate the sum of the previous two numbers. By saving only the last two sequence numbers, we can perform a loop to find additional Fibonacci sequences while never having to recalculate our previous positions.