Jonathan Simonin

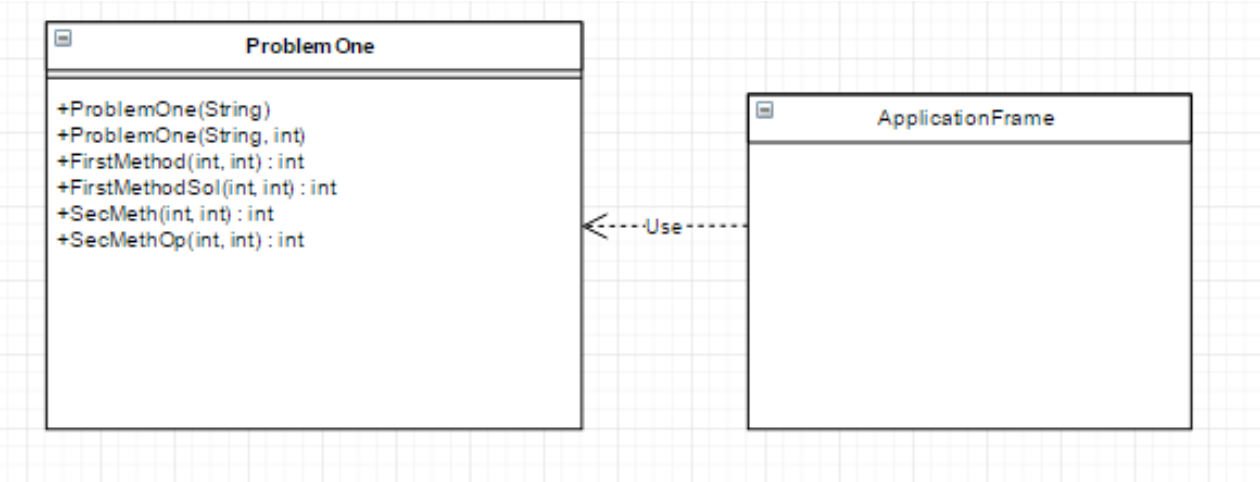
Professor Wei

CSE 2100

30 September 2016

Recursion Programming Assignment

**UML:**




**Analysis:**

If we take a close look at the graph of the first problem we can see that there is a y-asymptote at around y =26, so based on our data points we can see that it is getting close to approaching 26 calls per number at a large number. Based on the shape of the curve we can see that the Big Oh notation would be O(log(n)). (Refer to the screenshot)

As for the second problem, we cannot analyze the runtime as it follows no actual trend it seems. In fact, the assignment says that it is impossible to analyze as of right now because nobody has been able to prove this yet.

**Description:**

The overall program design is to mimic two recursive function given certain conditions, and then replicate them in a way that makes them more efficient – by either having fewer calls to the function and/or reducing the amount of statements or actions the method performs overall. The way I did this for the first problem: g(n) = 1 + g(n/2), if n is even; 1 + g(n-1), if n is odd and n> 1; 1, if n = 1 was by combining the return functions. When determining if the number given was odd, I would instead take g(n-1) and then divide that by two: g((n-1)/2) – thus creating a more efficient call system by performing two steps in one, thus calling the function one time less for each odd encounter it faced. Similarly, this was possible for the second function we had to

optimize: g(n) = 1 + g(n/2), if n is even; 1 + g(3n+1), if n is odd and n > 1; 1, if n =1. Instead of returning 1 + g(3n+1), I would return 1 + g((3n+1)/2) – thus creating the same end result as the first method. This overall change and extremely simple optimization showed to reduce the calls and (assumed) runtime as well substantially. Both methods work by taking in a parameter n and then it runs through each condition as stated previously, and the optimized function has been explained previously as well. The end result takes that original parameter down to 1, and once it reaches 1 it stops the recursion.

**Tradeoffs:**

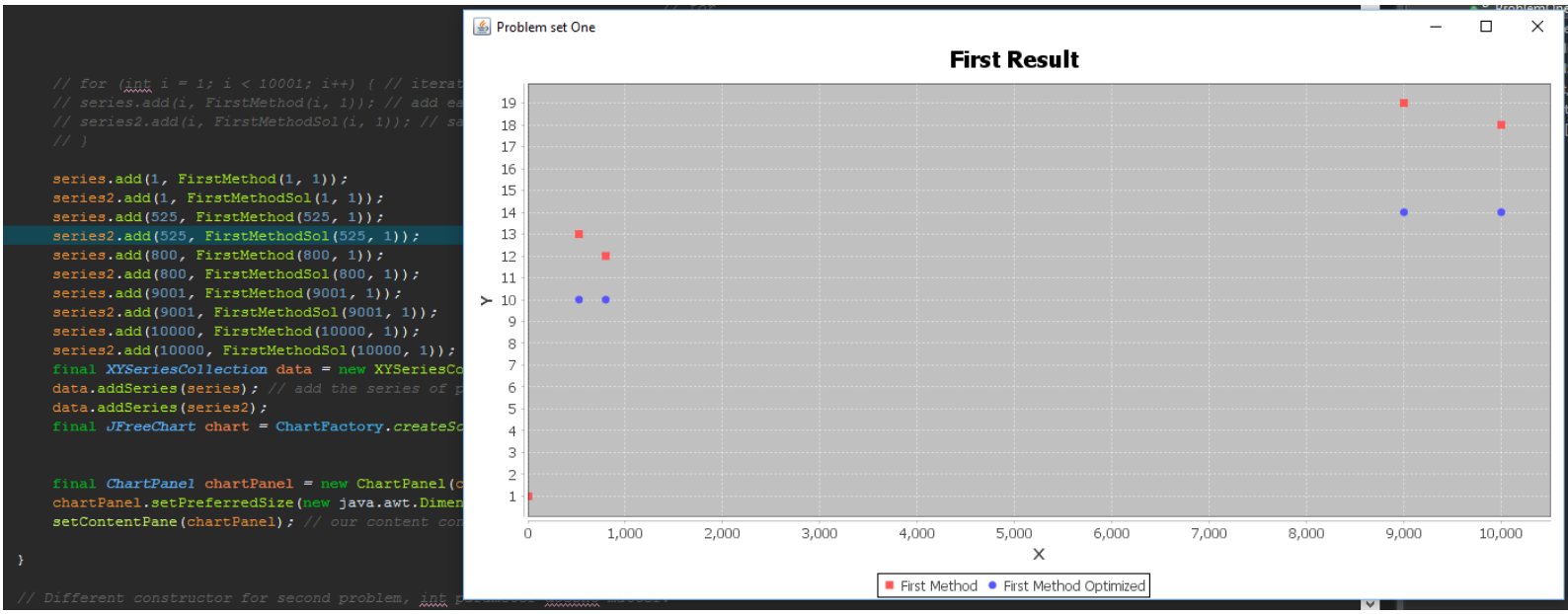
In this assignment I actually made no tradeoffs. I had met with a TA and he told me what he assumed the goal of what we were doing was, and once he explained it to me I ended up doing the solution he informally suggested. The end result was to combine what the function's return methods were in one whole step instead of making it separate every time. However, my original idea of how we had to optimize the program was to just create fewer recursive calls and that's it, but as mentioned above, the TA explained that this was not the case.

**Extensions:**

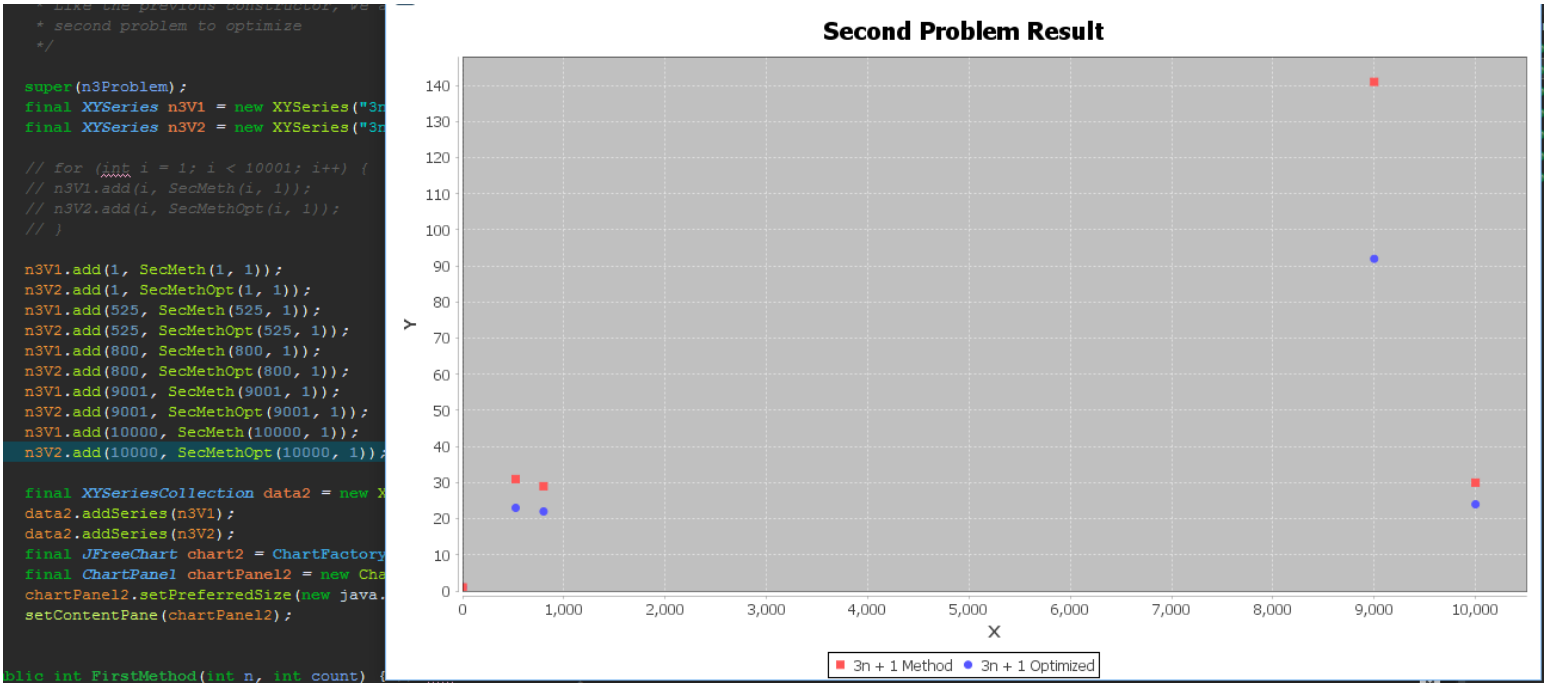
Possible improvements and extensions of this program could possibly be removing the restriction of optimizing both the runtime and recursive calls. If this was the case I planned on creating a for loop if the number was even, and I would devise it by the largest number it could divide into without a remainder, and going through the method again until it reached one. I had done this originally and it proved to only call the methods at max 4 times or so. Given our conditions on what to do with this assignment, I'm not exactly sure what other improvements I could possibly make – other than making the overall program more neat and organized.

**(Test cases on next page)**

**Test Cases:**




The test cases I chose were the values 1, 525, 800, 9001, 10000. I choose each of these for a certain reason. I checked 1 to see if it only went through 1 loop, which it does as shown by the graph. The reason I chose both 525 and 800 was because they were smaller numbers, one of them was odd and one was even. This way I could see if the optimization would work for both



even and odd numbers, which as shown by the graph it does. I chose 9001 and 10000 for the same exact reasons as the first pair of even and odd numbers tested, except it was to test if the optimization was working for both the larger odd and larger even numbers – and as shown it does. These same exact test cases were chosen for the second problem to optimize, and shown below is the graph and points plotted to display the results. Because each method only takes in an integer, it refuses to take characters or strings in – and will not run if they are entered so these test cases cannot be considered.