605.420: Algorithms for Bioinformatics

W.T. Door

JHU ID: 123456

Project 0 Analysis

Due Date: September 9, 2013

Dated Turned In: September 10, 2013

Project 0 Analysis

Analysis of Horner's Rule

The analytical description of Horner's Rule is attached. The source code that implements this runs in $\Theta(n)$ time, as determined by the analytical reasoning. Figure 1 shows the results of the code execution, with the R2 value demonstrating a very close fit to a straight line.

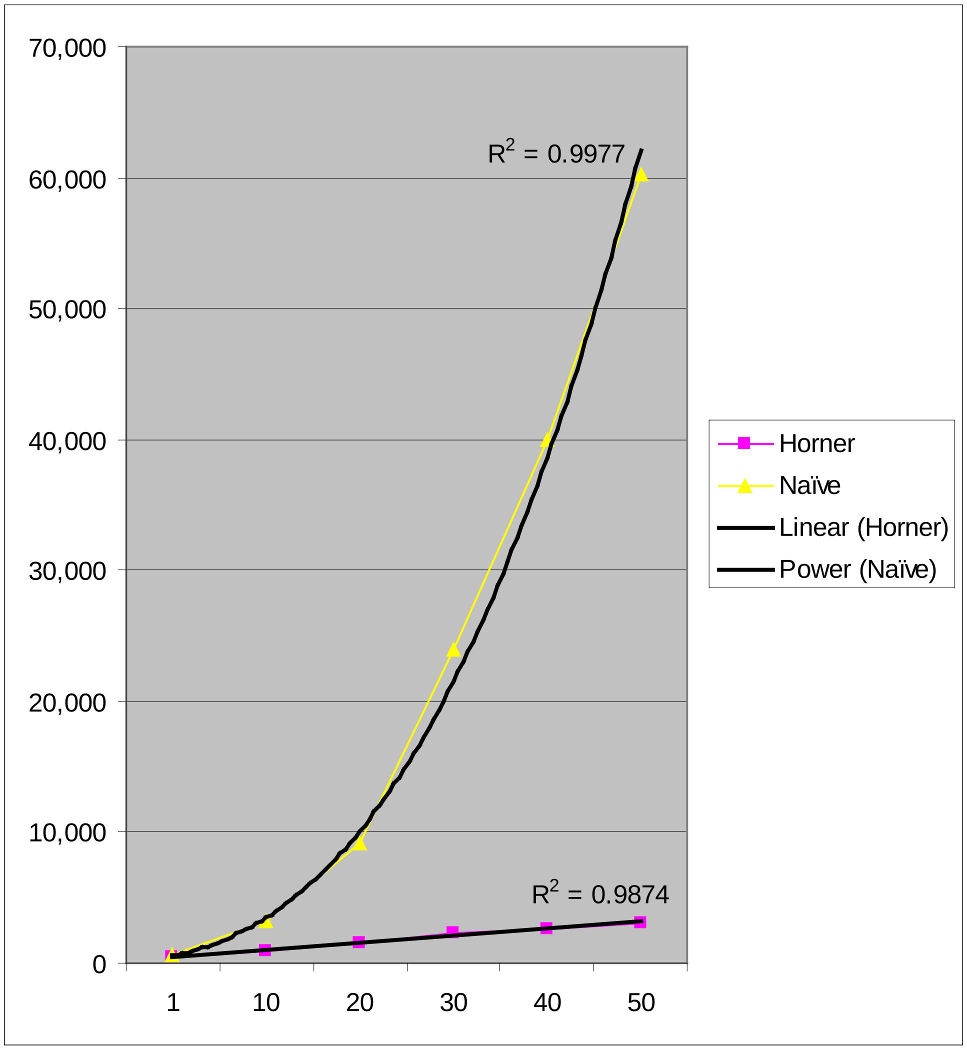

Figure 1: Graph of Results

Analysis of Naïve Polynomial

The analytical description of the Naïve Polynomial is attached. The source code that implements this runs in $\Theta(n^2)$ time, as determined by the analytical reasoning. Figure 1 shows the results of the code execution, with the R2 value demonstrating a very close fit to a power line.

Analysis of Loop Invariant

The analytical description of the Loop Invariant is attached. The computational output verifies the correctness of the Loop Invariant.

Verification that Horner's Rule Correctly Evaluates a Polynomial

The analytical description of the Horner's Rule correctness is attached. The computational output verifies this correctness in two ways.

First, the program outputs of Part a and Part b show that the *y* value calculated is the same for both Horner's Rule and the Naïve Polynomial. It is interesting that there are differences in the lower significant figures of the results.

Second, the Loop Invariant is shown to be correct for initialization, maintenance, and termination.

What I Learned

The values of $\Theta(\ )$ can actually be very accurate predictors of how an algorithm can perform. I was not expecting such a close correlation to curve fitting for these outputs.

Even on a personal computer with no other programs active, the timing results can vary. I used an average over ten repeats to get rid of some of this, but the results still vary a little bit.

The difference between $\Theta(n)$ and $\Theta(n^2)$ makes a very big difference after only a small increase in *n*. The naïve polynomial took over *20 times* as long with only 50 terms.

What I Might do differently Next Time

Write the program in C++. I had a really hard time figuring out how to do the input and output with Java and writing it in a language I know would have saved a lot of time.

The main driver class is pretty large compared to the other classes. The file reading and parsing routines could be made separate so that different file formats could be read easily instead of changing the main driver class.

Justification for Design Decisions

It seemed better to compare parts A and B directly, so I wrote the driver to solve both of these at once.

I wanted to see how the program was going directly, so I had the output print on the console instead of to a file. Then I just copied and pasted the output to a separate answer file.

Issues of Efficiency

Horner’s Rule used much fewer computing resources than a naïve solution. This is definitely a more efficient way of calculating polynomials.