4. Consider the problem of evaluating a polynomial at a point. Given n coefficients $a_0, a_1, ..., a_n$ and a real number x, we wish to compute $\sum_{ii=o}^{n-1} a_i x_i$. Describe a straightforward $\Theta(n^2)$-time algorithm for this problem. Describe a $\Theta(n)$-time algorithm that uses the following method (called Horner's rule) for rewriting the polynomial:

$$\sum_{i=0}^{n-1} a_i x^i = (...(a_{n-1}x + a_{n_2})x + ... + a_1)x + a_0$$

Computation of polynomials requires multiplication and addition. Let us assume, for our model of computation, that each multiplication and addition costs a constant amount of time, c_m and c_a respectively.

The computation of x^i is bounded from above by $\Theta(n)$ (i.e., a maximum of n multiplications). There are a total n such terms in the expression. Hence we should be able to evaluate the expression in $\Theta(n^2)$ time, in the worst case.

An example algorithm is as follows:

```
1 import java.util.List;
```

5. Express the function $n^3/1000 - 100n^2 - 100n + 3$ in terms of Θ-notation.

For a polynomial function f in n, only the highest order term is relevant when considering order of growth statistics. More precisely, suppose that

$$f(n) = \sum_{i=0}^{k} a_i n^i = a_k n^k + a_{k-1} n^{k-1} + ... + a_0$$

$$\frac{1}{n^k} f(n) = a_k + \frac{1}{n} a_{k-1} + ... + \frac{1}{n^k} a_0$$

so that

$$\lim_{n \to \infty} \frac{1}{n^k} f(n) = a_k$$

Hence for large n we can write

$$f(n) \approx a_k n^k$$

The constant coefficient a_k can be ignored, giving $\Theta(f(n)) = n^k$.

In the example above, where $f(n) = n^3/1000 - 100n^2 - 100n + 3$, even though the quadratic and linear terms have large negative coefficients and

even though the coefficient on the n^3 term is a small fraction, we can still write $\Theta(f(n)) = n^3$.

6. How can we modify almost any algorithm to have a good best-case running time?

Working..