COP 3503 Homework #5: Polynomial Multiplication

Filename: poly.java
Time Limit: 15 seconds (per input case)
Standard Input, Standard Output

In class, we learned Karatsuba's algorithm for multiplying two n-digit integers, where n is a perfect power of 2 in $O(n^{\log_2 3})$ time. This algorithm can be very slightly modified to work for polynomials of degree n, where n is a perfect power of 2 minus 1. For this assignment, you will implement Karatsuba's algorithm and be required to implement several methods. In addition, your code should process the input given, calling the appropriate method of the required ones and output the result of the computation in the desired output format.

The Problem:

Given two polynomials, calculate the product of the two polynomials via Karatsuba's algorithm and output the result.

The Input:

The first line of input contains a single integer, ***n*** ($0 \le \boldsymbol{n} \le 19$)**,** representing that there will be two polynomials of degree $\mathbf{2^n - 1}$ to be multiplied for the input case.

The second line of input contains $\mathbf{2^n - 1}$ space separated integers, representing the coefficients of the first polynomial in order from largest to smallest term. For example, the polynomial $5x^3 - 7x^2 + 8x + 6$ would be represented as

```
5 -7 8 6
```

The third line of input contains $\mathbf{2^n - 1}$ space separated integers, representing the coefficients of the second polynomial in order from largest to smallest term.

Each of the coefficients of both polynomials have an absolute value of 10^6 or less.

Note: This means that the product polynomial may have terms that only fit into a long but not an int.

The Output:

Output $\mathbf{2^{n+1} - 1}$ lines, each with a single coefficient of the product polynomial. The first line should have the coefficient to the $x^{2^{n+1}-2}$ term, the second line should have the coefficient to the $x^{2^{n+1}-3}$ term, …, and the last line should have the constant coefficient.

Sample Input **Sample Output**

Sample Input	Sample Output

2
5 -7 8 6
2 3 0 4
	10
	1
	-5
	56
	-10
	32
	24

1
3 1
2 7
	6
	23
	7

3
8 -7 3 6 1 15 2 -8
4 -2 -3 -4 -5 1 9 2
	32
	-44
	2
	7
	-29
	71
	1
	-159
	-36
	2
	58
	179
	40
	-68
	-16

Implementation Requirements

You must store a polynomial as an array of long where index i of the array stores the coefficient to the term x^i in the corresponding polynomial.

To encourage good object-oriented design, you will be required to build polynomial class with the two following instance variables:

```
private int length;
private long[] coeff;
```

where length is the length of the array coeff, and coeff is the array storing the coefficients of the polynomial.

Please implement the following methods:

```
// Creates a polynomial from the coefficients stored in vals.
// The polynomial created must store exactly (1<<k) coefficients
// for some integer k.
public poly(long[] vals);

// Both this and other must be of the same size and the
// corresponding lengths must be powers of 2.
// Returns the sum of this and other in a newly created poly.
public poly add(poly other);

// Both this and other must be of the same size and the
// corresponding lengths must be powers of 2.
// Returns the difference of this and other in a new poly.
public poly sub(poly other);

// Both this and other must be of the same size and the
// corresponding lengths must be powers of 2.
// Returns the product of this and other in a newly created
// poly, using the regular nested loop algorithm, with the
// length being the next power of 2.
public poly multSlow(poly other);

// Both this and other must be of the same size and the
// corresponding lengths must be powers of 2.
// Returns the product of this and other in a newly created
// poly, using Karatsuba's algorithm, with the
// length being the next power of 2.
public poly mult(poly other);

// Returns the left half of this poly in its own poly.
private poly getLeft();

// Returns the right half of this poly in its own poly.
private poly getRight();
```

When implementing Karatsuba's algorithm, use a base case of size 32 (1<<5) for this poly and other. In the base case, find the product by using the method multSlow, which runs the usual O(n^2) algorithm.

Also, since output could take a while, please use a StringBuffer or StringBuilder to store the output and then print all the output with a single System.out.print().

Deliverables

Please submit a single file called **poly.java**, which includes all of the instance methods of the poly class described, as well as a main method which processes input from standard input and outputs the result to standard output. Submit the file via Webcourses in the appropriate assignment submission.