

KING SAUD UNIVERSITY
COLLEGE OF COMPUTER & INFORMATION SCIENCES
DEPT OF COMPUTER SCIENCE

CSC311 Computer Algorithms
Second Semester 1427/1428 AH
Term Project Due:
Instructor:

TBA
Dr. Aqil Azmi

Fast Polynomial Multiplication

Introduction

Suppose that we have two polynomials $P = \sum_{i=0}^{n-1} p_i x^i$, and $Q = \sum_{i=0}^{n-1} q_i x^i$, both of degree $n-1$.

Then their product is

$$\begin{aligned} PQ &= \left(\sum_{i=0}^{n-1} p_i x^i \right) \left(\sum_{i=0}^{n-1} q_i x^i \right) \\ &= p_0 q_0 + \cdots + (p_{n-1} q_{i+1} + p_{n-2} q_{i+2} + \cdots + p_{i+1} q_{n-1}) x^{n+i} + \cdots + p_{n-1} q_{n-1} x^{2n-2}. \end{aligned}$$

The above multiplication requires $O(n^2)$ multiplications and additions. In order to achieve a cheaper scheme we will try divide & conquer technique as follows. For simplicity assume that n is a power of 2. Dividing each polynomial into two equal-sized parts, let $P = P_1 + P_2 x^{n/2}$, and

$Q = Q_1 + Q_2 x^{n/2}$, where P_1, P_2, Q_1, Q_2 are the polynomials

$$P_1 = p_0 + p_1 x + \cdots + p_{n/2-1} x^{n/2-1}, \quad P_2 = p_{n/2} + p_{n/2+1} x + \cdots + p_{n-1} x^{n/2-1},$$

and

$$Q_1 = q_0 + q_1 x + \cdots + q_{n/2-1} x^{n/2-1}, \quad Q_2 = q_{n/2} + q_{n/2+1} x + \cdots + q_{n-1} x^{n/2-1},$$

We now have

$$\begin{aligned} PQ &= (P_1 + P_2 x^{n/2})(Q_1 + Q_2 x^{n/2}) \\ &= P_1 Q_1 + (P_1 Q_2 + P_2 Q_1) x^{n/2} + P_2 Q_2 x^n. \end{aligned}$$

The recurrence relation describing the above algorithm is $T(n) = 4T(n/2) + O(n) = O(n^2)$. Still *no* improvement. Recall the trick we used in large integer multiplication where we reduced the number of multiplications to 3. That is, it suffices to compute the polynomials $P_1 Q_1$, $P_2 Q_2$ and $(P_1 + P_2)(Q_1 + Q_2)$. The middle term is just $(P_1 + P_2)(Q_1 + Q_2) - P_1 Q_1 - P_2 Q_2$. The new cost of polynomial multiplication is now $T(n) = O(n^{\log 3}) = O(n^{1.59})$.

For example, consider two polynomials $P = 1 - x + 2x^2 - x^3$, and $Q = 2 + x - x^2 + 2x^3$. We will compute their product using divide and conquer algorithm. Doing a single recursion step, then $P_1 = 1 - x$, $P_2 = 2 - x$, $Q_1 = 2 + x$, and $Q_2 = -1 + x$. Now observe that $P = P_1 + P_2 x^2 = (1 - x) + (2 - x)x^2$, and that $Q = Q_1 + Q_2 x^2 = (2 + x) + (-1 + 2x)x^2$. Computing $P_1 Q_1 = (1 - x)(2 + x) = 2 - x - x^2$, $P_2 Q_2 = (2 - x)(-1 + 2x) = -2 + 5x - 2x^2$, and

$(P_1 + P_2)(Q_1 + Q_2) = (3 - 2x)(1 + 3x) = 3 + 7x - 6x^2$. And so the middle term is $3 + 3x - 3x^2$. So the product of the polynomials is $PQ = (2 - x - x^2) + (3 + 3x - 3x^2)x^2 + (-2 + 5x - 2x^2)x^4$. Note that in real life situation we have to repeat the same process when computing the subproducts, i.e. P_1Q_1, P_2Q_2 and $(P_1 + P_2)(Q_1 + Q_2)$.

Project

Your task is to implement the above polynomial multiplication algorithm. In the above discussion we assumed that both polynomials are of degree which is a power of 2. In your implementation you have to work out the details if the polynomials are of unequal degree which is not a power of 2. At each recursion level, the program must print the input and the output polynomials. Following some further detail of what I would like to have.

- You may consult books and/or the web for ideas only and it must be duly acknowledged. Contact with other fellow humans is *not* allowed. The project write-up and the coding must be your own.
- The cover sheet will have your name(s) and a sign pledge indicating that the work has been done exactly according to the said specification.

What to Submit

- Cover sheet with your name(s) and the signed pledge.
- Write-up of the project (a brief description of your implementation, *i.e.* what kind of ADT used; cost analysis; sample runs and the conclusion).
- For sample run, execute your program on the polynomials below:
 - $P = 1 - x - 2x^2 - x^3, Q = 2 + x - x^2 + 2x^3$.
 - $P = 1 + x + x^2, Q = 1 + x + x^2 + x^3 + x^4$.
 - $P = 1 - 2x + 3x^3 - 4x^4, Q = 5 - 6x + 7x^2 - 8x^3 - 9x^4 + 10x^5 - 11x^6 + 12x^7 - 13x^8$
 - $P = 1 + x^6, Q = 2 - x^3 + x^{10}$.
- Floppy with the source list of the code and the executable.