Divide and Conquer: Karatsuba's Polynomial Multiplication[1]

In this lecture we look at a really fascinating application of the divide-and-conquer paradigm. The problem is that of multiplying two univariate polynomials.

Recall, given a variable x, a degree n polynomial $p(x)$ is of the form

$$p(x) = \sum_{i=0}^{n} p_i \cdot x^i$$

where p_i is the *coefficient* of the *degree i monomial* x^i. A degree n polynomial has $(n+1)$ monomials (including the constant monomial $x^0 = 1$) and coefficients.

Given two degree n polynomials, $p(x)$ and $q(x)$, the ***product*** of the two polynomials $p(x) \cdot q(x)$ is *another* polynomial $r(x)$. Let us recall this with an example. Consider

$$p(x) = 1 + x + x^2 \quad \text{and} \quad q(x) = 2 + 3x + x^2$$

Then, the product polynomial is

$$r(x) \;=\; (1 + x + x^2)(2 + 3x + x^2) \;=\; 2 + 5x + 6x^2 + 4x^3 + x^4$$

Indeed, in general, if $p(x)$ and $q(x)$ are degree n polynomials, then $r(x)$ is a degree $2n$ polynomial, whose coefficient r_k for the monomial x^k, $0 \le k \le 2n$ is given by the formula

$$r_k = \begin{cases} \sum_{0 \le i \le k} \; p_i \cdot q_{k-i} & \text{if } k \le n \\ \sum_{(k-n) \le i \le n} \; p_i \cdot q_{k-i} & \text{if } n < k \le 2n \end{cases} \tag{1}$$

For instance, $r_2 = p_0 q_2 + p_1 q_1 + p_2 q_0$.

MULTIPLYING POLYNOMIALS
Input: Coefficients of two degree n polynomials: arrays $P[0:n]$ and $Q[0:n]$
Output: Coefficients of the product polynomial: array $R[0:2n]$.
Size: n, the length of P and Q.

We also assume that every $P[i], Q[j]$ are "small" numbers ; they can be added and multiplied in $O(1)$ time.

An $O(n^2)$ time algorithm follows from the formula (1). Indeed, for every k, where $0 \le k \le 2n$, we need compute only a summation. The kth summation adds at most $(n+1)$ summands, and each summand is product of two numbers. The summands can be found using a for-loop taking $O(n)$ time. In sum, every $R[k]$, individually, can be computed in $O(n)$ time. Since there are $2n+1$ different k's, one can figure the whole $R[0:2n]$ out in $O(n^2)$ time.

How can we do better? Perhaps one thought that may come to you is the following: each individual $R[k]$ computation sums up many different products; perhaps these are shared by different k's? And if so, one probably doesn't need to recompute. Unfortunately, that is not the case. For example $R[2] = P[0]Q[2] + P[1]Q[1] + P[2]Q[0]$. But, $R[3] = P[0]Q[3] + P[1]Q[2] + P[2]Q[1] + P[3]Q[0]$. No summands are shared. Bummer!

[1] *Lecture notes by Deeparnab Chakrabarty. Last modified : 19th Mar, 2022*
These have not gone through scrutiny and may contain errors. If you find any, or have any other comments, please email me at deeparnab@dartmouth.edu. Highly appreciated!

Remark: *At this point, it is natural to probably say, "Maybe one cannot do any better." And if so, you are in venerable company. The story goes that in the early 1960s the famous Russian mathematician Andrei Kolmogorov held a seminar with the objective to show that* ***any*** *algorithm must need $\Omega(n^2)$ time to multiply two degree n polynomials. After the first meeting, a young student named* ***Anatoly Karatsuba*** *came up with the algorithm we are about to describe. Kolmogorov canceled the remainder of the seminar. Like all good stories, this is probably untrue.*

And the algorithm is a simple, but magical, divide-and-conquer algorithm. Let's begin.

Remark: *It may be useful to keep a "running example" to illustrate the algorithm. So, suppose our example (for $n = 3$) is*

$$\mathbf{p}(x) = 1 + 3x + x^2 + 2x^3 \quad \textit{and} \quad \mathbf{q}(x) = 2 + x + 2x^2 + x^3$$

The product polynomial is

$$\mathbf{r}(x) = 2 + 7x + 7x^2 + 12x^3 + 7x^4 + 5x^5 + 2x^6$$

The boldface is just to make you aware that it is a specific example.

We will start with an algorithm which *doesn't* quite do the job, and then fix it. Let $m = \lceil n/2 \rceil$. Consider the polynomial $p(x)$ and write it as

$$p(x) = p_1(x) + x^m p_2(x) \quad \text{where} \quad p_1(x) = \sum_{i=0}^{m-1} P[i]x^i \quad \text{and} \quad p_2(x) = \sum_{i=0}^{n-m} P[m+i]x^i \tag{2}$$

Similarly write

$$q(x) = q_1(x) + x^m q_2(x) \quad \text{where} \quad q_1(x) = \sum_{j=0}^{m-1} Q[j]x^j \quad \text{and} \quad q_2(x) = \sum_{j=0}^{n-m} Q[m+j]x^j \tag{3}$$

Note that all four polynomials $p_1(x), p_2(x), q_1(x), q_2(x)$ have degree $\leq \lfloor n/2 \rfloor$. For our example, we have $m = \lceil 3/2 \rceil = 2$, and thus

$$\mathbf{p}_1(x) = 1 + 3x, \quad \mathbf{p}_2(x) = 1 + 2x, \quad \mathbf{q}_1(x) = 2 + x, \quad \mathbf{q}_2(x) = 2 + x$$

Now, we see that the product $r(x)$ of $p(x)$ and $q(x)$ can be written thus:

$$\begin{aligned} r(x) &= (p_1(x) + x^m p_2(x)) \cdot (q_1(x) + x^m q_2(x)) \\ &= \Big(p_1(x) \cdot q_1(x)\Big) + x^m \cdot \Big(p_1(x) \cdot q_2(x) + p_2(x) \cdot q_1(x)\Big) + x^{2m} \cdot \Big(p_2(x) \cdot q_2(x)\Big) \end{aligned} \tag{4}$$

Therefore, (4) implies that $r(x)$ can be computed by ***recursively*** multiplying the four pairs of polynomials $(p_1(x), q_1(x))$, $(p_1(x), q_2(x))$, $(p_2(x), q_1(x))$, and $(p_2(x), q_2(x))$. Each pair is a product of polynomials of degree at most $\lfloor n/2 \rfloor$. After computing these four products, we need to *add* these four product polynomials up. This is the "conquer/combine" step.

How much time does it take to add up two degree k polynomials? Let us figure this out. Given two degree n polynomials, let us now call them $a(x)$ and $b(x)$, the addition is another degree n polynomial whose kth coefficient is simply the sum of the corresponding kth coefficients of $a(x)$ and $b(x)$. Thus, one can obtain the sum of two degree n polynomials in $O(n)$ time.

To summarize, the suggested recursive algorithm is to compute four products: (1) $r_1(x) = p_1(x)q_1(x)$, $r_2(x) = p_1(x)q_2(x)$, $r_3(x) = p_2(x)q_1(x)$, and $r_4(x) = p_2(x)q_2(x)$ recursively. And then, outputting $r(x) = r_1(x) + x^m \cdot (r_2(x) + r_3(x)) + x^{2m} r_4(x)$. Note that $x^{2m} r_4(x)$ is simply another polynomial whose coefficients are "shifted" by $2m$. The following pseudocode gives the outline (but I am not providing details).

```
1: procedure MULTPOLYDC(p(x), q(x)):▷ We want to return p(x) · r(x).
2:     m ← ⌈n/2⌉
3:     Form the polynomials p_1(x), p_2(x), q_1(x), q_2(x) respectively. ▷ This takes O(n) time.
4:     r_1(x) ←MULTPOLYDC(p_1, q_1) ▷ This takes T(⌈n/2⌉) time.
5:     r_2(x) ←MULTPOLYDC(p_1, q_2) ▷ This takes T(⌈n/2⌉) time.
6:     r_3(x) ←MULTPOLYDC(p_2, q_1)▷ This takes T(⌈n/2⌉) time.
7:     r_4(x) ←MULTPOLYDC(p_2, q_2)▷ This takes T(⌈n/2⌉) time.
8:     Form r(x) by combining r_1(x), r_2(x), r_3(x), r_4(x). ▷ This takes O(n) time since adding polynomials takes O(n) time.
```

Just to illustrate, for our example polynomials, we get that

$$\mathbf{r}_1(x) = 2 + 7x + 3x^2, \quad \mathbf{r}_2(x) = 2 + 7x + 3x^2, \quad \mathbf{r}_3(x) = 2 + 5x + 2x^2, \quad \mathbf{r}_4(x) = 2 + 5x + 2x^2,$$

And therefore, the algorithm would return the polynomial

$$(2 + 7x + 3x^2) + x^2 \left((2 + 7x + 3x^2) + (2 + 5x + 2x^2)\right) + x^4 \left(2 + 5x + 2x^2\right)$$

which equals

$$2+7x+3x^2+\left(4x^2+12x^3+5x^4\right)+\left(2x^4+5x^5+2x^6\right) = 2+7x+7x^2+12x^3+7x^4+5x^5+2x^6 \quad (5)$$

which is what it should be (that is, $\mathbf{r}(x)$.).

What is the running time of the above algorithm? Well, it breaks a problem into *four* subproblems each of size $\lfloor n/2 \rfloor$ and then combines them in time $O(n)$. That is, the recurrence inequality governing the running time is

$$T(n) \leq 4T(\lfloor n/2 \rfloor) + O(n)$$

We apply the Master Theorem, and then we get $T(n) = O(n^2)$. Sigh! Much ado about nothing?

Next comes the Aha! insightful observation. We observe that we really don't need the individual products $p_1(x) \cdot q_2(x)$ and $p_2(x) \cdot q_1(x)$ at all. What we need is just their sum. Can we compute the sum *without* computing the individual summands? The answer is yes! It follows from the following trivial but key observation.

Observation 1.

$$p_1(x)q_2(x) + p_2(x)q_1(x) = \Big(p_1(x) + p_2(x)\Big) \cdot \Big(q_1(x) + q_2(x)\Big) - \Big(p_1(x) \cdot q_1(x)\Big) - \Big(p_2(x) \cdot q_2(x)\Big)$$

Proof. Just open up the brackets and see. □

Again going back to our example, we see that

$$(\mathbf{p}_1(x) + \mathbf{p}_2(x)) \cdot (\mathbf{q}_1(x) + \mathbf{q}_2(x)) = (2 + 5x) \cdot (4 + 2x) = (8 + 24x + 10x^2)$$

And thus,

$$\mathbf{r}_2(x) + \mathbf{r}_3(x) = (8 + 24x + 10x^2) - (2 + 7x + 3x^2) - (2 + 5x + 2x^2) = 4 + 12x + 5x^2$$

which is indeed the case. And as in (5), we proceed to get the right product of $\mathbf{p}(x)$ and $\mathbf{q}(x)$.

Why is this observation useful? Well, note that $p_1(x)q_1(x)$ and $p_2(x)q_2(x)$ have been computed already (these are $r_1(x)$ and $r_4(x)$).

Therefore, to compute the sum in the LHS, that is $r_2(x) + r_3(x)$, we don't have to compute them individually, but rather compute the product $(p_1(x) + q_1(x)) \cdot (p_2(x) + q_2(x))$ and subtract the $r_1(x)$ and $r_4(x)$ from this. Thus, we can get away with *three* multiplications of smaller polynomials.

procedure KARATMULTPOLY($p(x), q(x)$):▷ *We want to return $p(x) \cdot r(x)$.*
    $m \leftarrow \lceil n/2 \rceil$
    Form the polynomials $p_1(x), p_2(x), q_1(x), q_2(x)$ respectively. ▷ *This takes $O(n)$ time.*
    $r_1(x) \leftarrow$KARATMULTPOLY(p_1, q_1) ▷ *This takes $T(\lfloor n/2 \rfloor)$ time.*
    $r_4(x) \leftarrow$KARATMULTPOLY(p_2, q_2) ▷ *This takes $T(\lfloor n/2 \rfloor)$ time.*
    Compute polynomials $p'(x) = p_1(x) + p_2(x)$ and $q'(x) = q_1(x) + q_2(x)$.▷ *This takes $O(n)$ time since adding polynomials takes $O(n)$ time.*
    $s(x) \leftarrow$KARATMULTPOLY(p', q')▷ *This takes $T(\lfloor n/2 \rfloor)$ time.*
    $t(x) \leftarrow s(x) - r_1(x) - r_4(x)$. ▷ *This takes $O(n)$ time since adding/subtracting polynomials takes $O(n)$ time.*
    Form $r(x)$ by combining $r_1(x), r_4(x), t(x)$. More precisely, $r(x) = r_1(x) + x^m \cdot t(x) + x^{2m} \cdot r_4(x)$. ▷ *This takes $O(n)$ time since adding polynomials takes $O(n)$ time.*

One can now see that the recurrence inequality governing the above algorithm becomes

$$T(n) \leq 3T(\lceil n/2 \rceil) + \Theta(n)$$

which gives us the following.

Theorem 1. The algorithm KARATMULTPOLY multiplies two n-degree univariate polynomials in $O(n^{\log_2 3}) = O(n^{1.59})$ time.

Below, we give another pseudocode which considers the input as arrays of the coefficients. This may help you in actually coding it up. Indeed, you this will be asked in the coding assignment.

procedure KARATMULTPOLY($P[0:n], Q[0:n]$):▷ *We want to return* $R[0:2n]$.
  if $n = 0, 1$ **then**:
    return $R[0:2n]$ using the naive multiplication
  $m = \lceil n/2 \rceil$.
  ▷ *Recall definitions of* $p_1(x), p_2(x), q_1(x), q_2(x)$ *from* (2),(3)
  for $0 \leq i \leq m-1$ **do**
    $P'[i] = (P[i] + P[m+i])$
    $Q'[i] = (Q[i] + Q[m+i])$
  if $n > 2m-1$ **then**: ▷ *In which case* $n = 2m$ *since* $m = n/2$ *or* $m = (n+1)/2$.
    $P'[m] = P[n]$
    $Q'[m] = Q[n]$
  else:
    $P'[m] = Q'[m] = 0$
  ▷ *Now* P' *has the coefficients of* $p_1(x) + p_2(x)$. Q' *has the coefficients of* $q_1(x) + q_2(x)$.
  ▷ *Their degrees are* $m-1$ *or* m *depending on the parity of* n.
  ▷ *The else statement above forces degree* m.

  $R_1[0:2(m-1)] =$ KARATMULTPOLY $(P[0:m-1], Q[0:m-1])$
  $R_2[0:2(n-m)] =$ KARATMULTPOLY $(P[m:n], Q[m:n])$
  $R_3[0:2m] =$ KARATMULTPOLY $(P'[0:m], Q'[0:m])$
  ▷ R_1 *has the coefficients of* $p_1(x) \cdot q_1(x)$
  ▷ R_2 *has the coefficients of* $p_2(x) \cdot q_2(x)$
  ▷ R_3 *has the coefficients of* $(p_1(x) + p_2(x)) \cdot (q_1(x) + q_2(x))$
  ▷ *Also note that* R_1, R_2, R_3 *all have length* $\leq 2m$. *We assume they all are* $2m$ *length by padding* 0*'s.*
  for $0 \leq i \leq 2m$ **do**:
    $R_4[i] = (R_3[i] - R_1[i] - R_2[i])$
  ▷ R_4 *has the coefficients of* $p_1(x) \cdot q_2(x) + p_2(x) \cdot q_1(x)$ *and is degree* $2m$
  for $0 \leq i \leq 2n$ **do**:
    $R[i] = R_1[i] + R_4[i-m] + R_2[i-2m]$
    ▷ *We assume an array 'returns* 0*' if indexed out of its range. For instance,* $R_4[-1]$ *returns* 0 *and* $R_1[2n]$ *returns* 0.
    ▷ *When you actually code it, you need a few "if" statements to implement the above. Please do that – it's super instructive.*
  return $R[0:2n]$