**E0 309: Topics in Complexity Theory** **Spring 2015**

Lecture 1: Jan 14, 2015

*Lecturer: Neeraj Kayal* *Scribe: Sumant Hegde and Abhijat Sharma*

## 1.1 Introduction

The theme of the course in this semester is Algebraic Complexity Theory with primary focus on arithmetic circuit complexity. At a very high level, this area studies complexity of arithmetic or algebraic operations by 'translating' problems to computation of/with polynomials. This in a way is a relatively structured approach to computational problems, as it is more concerned with *syntactic computation* as compared to *semantic computation*, like in the boolean setting[1].

Thinking of algebraic problems, the first two basic operations are addition and multiplication. In the case of addition of two integers the middle school algorithm performs $O(n)$ bit-by-bit additions which is optimal. In the case of multiplication of two integers the middle school algorithm performs $O(n^2)$ operations. But the optimal complexity of multiplication of two integers is still an open problem.

### 1.1.1 Karatsuba's algorithm(1962)

It was conjectured (by Kolmogorov) that the naive multiplication algorithm was asymptotically optimal[6]. But this turned out to be false due to the discovery of Karatsuba's algorithm which we discribe now. Given two $n$-digit numbers $A$ and $B$, suppose we split them into blocks $A_0, A_1$ and $B_0, B_1$ as follows.

$$A = A_0 + 10^{n/2} A_1$$
$$B = B_0 + 10^{n/2} B_1$$

Then the product $AB$ can be expressed as,

$$AB = A_0B_0 + 10^{n/2}(A_0B_1 + A_1B_0) + 10^n A_1B_1$$

We could recursively compute each of the four products to get $AB$. Unfortunately, this approach is asymptotically no better than the previous one:

$$\begin{aligned} T(n) =& 4T(n/2) + O(n) \triangleright \text{Here } O(n) \text{ is for additions and multiplications by 10's powers} \\ =& O(n^2) \end{aligned}$$

However, if we can somehow compute $A_0B_0$, $A_0B_1 + A_1B_0$, $A_1B_1$ with three (recursive) multiplication operations instead of four, then we get

$$\begin{aligned} T(n) =& 3T(n/2) + O(n) \\ =& O(n^{\log_2 3}) \end{aligned}$$

Karatsuba algorithm gives us a way of achieving this, as discussed below.
Assume additions and subtractions are completely free.

Input: $A_0, A_1, B_0, B_1$
Output: $A_0B_0, A_0B_1 + A_1B_0, A_1B_1$ $l_1 := A_0B_0$
$l_2 := (A_0 + A_1)(B_0 + B_1)$
$l_3 := A_1B_1$
$l_4 := l_2 - l_1 - l_3$
output $l_1, l_4, l_2$

Clearly, only 3 multiplications (namely, for $l_1, l_2, l_3$) are required. Hence the complexity of Karatsuba algorithm is $O(n^{\log_2 3})$ as discussed before.
Again, it is natural to ask whether we can do better. Before answering that question, we observe that integer multiplication can be translated to polynomial multiplication.

## 1.2 Polynomial multiplication

Input: Two univariate polynomials $A, B$:

$$A(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1}$$
$$B(x) = b_0 + b_1x + \cdots + b_{n-1}x^{n-1}$$

Output:

$$A(x)B(x) = (a_0b_0) + (a_0b_1 + a_1b_0)x + (a_0b_2 + a_1b_1 + a_2b_0)x^2 + \cdots + a_{n-1}b_{n-1}x^{2n-2}$$

**Assumptions:**
▷ +,- and multiplication by scalars are free.
▷ Each $\times$ has cost 1.

On putting $a_2 = a_3 = \cdots = a_{n-1} = 0$ and $b_2 = b_3 = \cdots = b_{n-1} = 0$, and evaluating $x$ at an appropriate power of 10, the model simplifies to Karatsuba's algorithm.

We can also represent an $n-1$-degree univariate polynomial $A$ as a list of evaluations:
$A = A(\alpha_1), \ldots, A(\alpha_n)$ where all $\alpha_i$'s are distinct.

**Fact 1:** There is a one-to-one correspondence between the two representations of $A(x)$, namely, as a list of $n$ coefficients and as a list of $n$ evaluations.
We identify by **evaluation** the process of getting the list of evaluations from the coefficient vector, and we identify by **interpolation** the other direction. We first detail the procedure of interpolation.

Given a polynomial $A(x)$ evaluated at points $\alpha_1, \ldots, \alpha_n$, suppose we need to find out its coefficients $a_0, \ldots, a_{n-1}$. We just need to solve the following system of equations (where $a_i$ are unknowns).

$$\begin{aligned}
A(\alpha_1) &= a_0 + a_1\alpha_1 + \cdots + a_{n-1}\alpha_1^{n-1} \\
A(\alpha_2) &= a_0 + a_1\alpha_2 + \cdots + a_{n-1}\alpha_2^{n-1} \\
&\vdots \\
A(\alpha_n) &= a_0 + a_1\alpha_n + \cdots + a_{n-1}\alpha_n^{n-1}
\end{aligned}$$

That is,

$$\underbrace{\begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \cdots & \alpha_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \cdots & \alpha_n^{n-1} \end{bmatrix}_{n\times n}}_{M} \cdot \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}_{n\times 1}}_{C} = \underbrace{\begin{bmatrix} A(\alpha_1) \\ A(\alpha_2) \\ \vdots \\ A(\alpha_n) \end{bmatrix}_{n\times 1}}_{E}$$

We can see that $M$ is a square Vandermonde matrix whose determinant is given by

$$DET(M) = \prod_{1\le j<i\le n} (\alpha_i - \alpha_j)$$

which can be proved by induction on $n$. This implies that $M$ is invertible (as $\alpha_i$'s are distinct), and hence we can compute the coefficient vector $C$, by $C = M^{-1}E$

Now we show that interpolation can be done in linear time (under the assumption that +,- and multiplication by scalars are free). $C$ can be viewed as a linear combination of columns of $M^{-1}$. But both $M$ and $M^{-1}$ can be precomputed as they are independent of the input. Under the assumption that additions and scalar multiplications are free, computing a linear combination takes constant time and thus $C$ can be computed in linear time.
□

**Upper bound** An important implication of Fact 1 is as follows. If two degree-$(n-1)$ polynomials $A, B$ are given as evaluations, i.e.

$$A = (A(\alpha_1), A(\alpha_2), \ldots, A(\alpha_{2n-1}))$$
$$B = (B(\alpha_1), B(\alpha_2), \ldots, B(\alpha_{2n-1}))$$

then the product polynomial can be given by the pointwise product of the evaluations, i.e.

$$AB = (A(\alpha_1)B(\alpha_1),\ A(\alpha_2)B(\alpha_2),\ \ldots,\ A(\alpha_{2n-1})B(\alpha_{2n-1}))$$

Therefore, the process of finding the product of two degree-$(n-1)$ polynomials $A(x), B(x)$ can be done in the following steps:
1. Evaluate $A(x), B(x)$ at $2n-1$ points.
2. Pairwise multiply to get the evaluations of $AB$ at $2n-1$ points.
3. Interpolate the coefficients of $AB$ from these evaluations.

This implies that, under the assumptions we made in the beginning of this section, $2n-1$ nonscalar multiplications are sufficient to multiply two degree-$(n-1)$ univariate polynomials.

## 1.3 Integer multiplication

Using the upper bound mentioned above, we are ready to answer whether we can do better than Karatsuba algorithm. This time let us split the input numbers into 4 parts:

$$\begin{aligned} A &= A_0 + A_1 10^{n/4} + A_2 10^{2n/4} + A_3 10^{3n/4} \\ B &= B_0 + B_1 10^{n/4} + B_2 10^{2n/4} + B_3 10^{3n/4} \\ AB &= (A_0B_0) + (A_0B_1 + A_1B_0)10^{n/4} + \cdots + (A_3B_3)10^{6n/4} \end{aligned}$$

This can be viewed as multplying two degree-3 polynomials. Therefore $n = 4$ and the upper bound above implies $2n - 1 = 7$ multiplication operations. Thus we get the recurrence

$$\begin{aligned} T(n) &= 7T(n/4) + O(n) \\ &= O(n^{\log_4 7}) \\ &< O(n^{\log_2 3}) \end{aligned}$$

This idea was used to further improve the complexity in Toom Cook algorithm, which splits up both the inputs into $k$ smaller parths and performs operations on the parts. As $k$ grows, one may combine many of the multiplication suboperations, thus reducing the overall complexity of the algorithm[2]. (The above example is just a special case of Toom Cook where $k = 4$).
Advanced algorithms like Schönhage–Strassen also use techniques like Fast Fourier Transforms (FFT) and modular arithmetic. Complexity of Schönhage-Strassen is $O(n \log n \log \log n)$. The best complexity known till date is $n \log n 2^{O(\log^* n)}$, achieved by Fürer(2007)[5] which uses FFT over complex numbers. De et al.(2008)[4] used modular arithmetic and achieved the same complexity. Recently, more explicit bound of $O(n \log n 8^{\log^* n})$ was shown by Harvey et al(2014)[3].

## 1.4 Lower bound on Polynomial Multiplication

In our quest to find the optimal way to multiply two univariate polynomials, we are trying to find the minimum number of *multiplication* operations required to compute the product of two polynomials, assuming the following operations as free : *addition*, *subtraction* and *multiplication by fixed constants*. We have already seen an upper bound on this quantity, now we try to find an answer for the coresponding lower bound.

We proceed with this lower bound proof using the concept of **Straight Line Programs (SLP)**. Given two univariate polynomials, each of degree $(n - 1)$, Consider the following SLP:

The first $2n$ lines are the coefficients of input polynomials

$$\begin{aligned} l_1 &= a_0 \\ l_2 &= a_1 \\ &\vdots \\ l_n &= a_{n-1} \\ l_{n+1} &= b_0 \\ &\vdots \\ l_{2n} &= b_{n-1} \end{aligned}$$

After the input, there are $s$ lines of intermediate computation, which involves multiplying appropriate linear combinations of the previous lines $(2n + 1 \leq i \leq 2n + s)$

$$l_i = \sum_{j<i} (\alpha_j l_j + c_j) \sum_{k<i} (\beta_k l_k + d_k)$$

where $c_j$ and $d_k$ are constants. And finally, there are the output lines, also computed as just a linear combination of all previous lines. each output on a seperate line, which in this case are the set of $(2n-1)$ coefficients of the product polynomial.

$$l_{2n+s+1} = \sum_{j<i} (\hat{\alpha_j} l_j) = a_0 b_0$$
$$l_{2n+s+2} = a_0 b_1 + a_1 b_0$$
$$\vdots$$
$$l_{2n+s+(2n-1)} = a_{n-1} b_{n-1}$$

Now, let us think of the coefficients of the input polynomials $(a_0, a_1, ...a_{n-1}, b_0, b_1, ..., b_{n-1})$ as variables. Then the outputs, that are the coefficients of the product polynomial, would be quadratic polynomials in the above defined variables. Every coefficient can now be written as a linear combination of monomials and therefore represented as a vector as follows. This is done after fixing some particular ordering of the $n^2$ possible monomials, $a_0b_0, a_0b_1, ..., a_{n-1}b_{n-1}$, and then representing each output as a vector, where each component of the vector corresponds to the coefficient of that particular monomial in the output. For example, the first output, $a_0b_0$ would be represented as the vector $[1, 0, 0, ..., 0]$, and the second output $a_0b_1 + a_1b_0$ will have two 1s in its vector representation, corresponding to the monomials $a_0b_1$ and $a_1b_0$, and the rest would be 0s.

After having represented each of the outputs as a vector as described above, consider the vector space spanned by these vectors over the field of real numbers, $\mathbb{R}$.

**Claim 1.1** *The dimension of the vector space, $dim_{\mathbb{R}}\{l_{2n+s+1}, ..., l_{2n+s+(2n-1)}\} \leq s$.*

**Proof:**

Let us think of all the computations of the straight line program as computing a polynomial in the variables $a_0, a_1, ...a_{n-1}, b_0, b_1, ..., b_{n-1}$. Then, the first $2n$ lines of the program are degree 1 polynomials, and every output is a degree 2 polynomial. Each of the output lines $l_{2n+s+1}, ..., l_{2n+s+(2n-1)}$ are linear combinations of the previous $(2n+s)$ lines. Thus, for $2n+s+1 \leq k \leq 2n+s+(2n-1)$,

$$l_k = \alpha_1 a_0 + \alpha_2 a_1 + ... + \alpha_{2n} b_{n-1} + \alpha_{2n+1} l_{2n+1} + ... + \alpha_{2n+s} l_{2n+s}$$

Now, let us restrict every line of the SLP to monomials of the form $a_ib_j$, then the above equaity should still hold for the restricted polynomials. As all the monomials in the output lines are of the form $a_ib_j$, the left hand side of the above equation stays as it is. Although, on the right hand side, lines $l_1, ...l_{2n}$ contain only monomials having degree 1, and therefore will be equal to zero when restricted to the degree-2 monomials of the form $a_ib_j$. Hence, in the resticted form, the output lines would look like

$$l_k = \alpha_{2n+1} \hat{l}_{2n+1} + ... + \alpha_{2n+s} \hat{l}_{2n+s}$$

where, $2n+s+1 \leq k \leq 2n+s+(2n-1)$ and $\hat{l_j}$ represent the polynomial $l_j$ restricted to monomials of the form $a_ib_j$. Now, we can see that all the $2n-1$ output lines are linear combinations of $s$ fixed polynomials, each represented as a vector of coefficients. Hence, the dimension of the vector space spanned by the vectors representing the output lines can be at-most $s$. This proves the claim that $dim_{\mathbb{R}}\{l_{2n+s+1}, ..., l_{2n+s+(2n-1)}\} \leq s$. ■

**Claim 1.2** $dim_{\mathbb{R}}\{a_0b_0, a_0b_1 + a_1b_0, ..., a_{n-1}b_{n-1}\} = 2n-1$

**Proof:** As the input polynomials are of degree $(n-1)$ each, the product is of degree $(2n-2)$ and hence will have $(2n-1)$ coefficients. Let us call the vector representations of these coefficients as $u_1, u_2, ..., u_{2n-1}$. Thus, to prove that the dimension of the spanned vector space is $2n-1$, it suffices to prove that all these $2n-1$ vectors are linearly independent.

We prove this by contradiction. Let us assume that they are not linearly independent, which implies that there exists some real numbers $\alpha_i (1 \leq i \leq 2n-1)$, not all zero such that $\sum_{i=1}^{2n-1} \alpha_i u_i = 0$. Now, we know that every component in these vectors corresponds to the coefficient of some particular monomial $a_i b_j$. On careful observation, we can notice that all these outputs are monomial disjoint, and therefore any particular monomial $a_i b_j$ occurs in exactly one of the $2n-1$ outputs. Also, when we compute the linear combination $\sum_{i=1}^{2n-1} \alpha_i u_i$, it will be zero if and only if all its components are zero. But, as every monomial occurs in only one output, its coefficient would not be cancelled out by any other vector representing another output. Hence, $\sum_{i=1}^{2n-1} \alpha_i u_i = 0$ iff $\alpha_i = 0$ for all $i$. This contradicts our assumption and therefore proves our claim.

■

From the above two claims, it directly follows that $s \geq 2n-1$. This proves that if we ignore all the addition and scalar multiplication operations, we still need at-least $2n-1$ multiplication operations to compute the product of two polynomials, each of degree $n-1$. This also shows that in the Karatsuba Algorithm, the breaking up of integers is like considering a degree 1 polynomial i.e $n = 2$. Therefore, it cannot be done in less than $2n - 1 = 3$ multiplications.

## 1.5 References


[1] Amir Shpilka and Amir Yehudayaoff, Arithmetic Circuits: A survey of recent results and open questions, *Foundations and Trends in Theoretical Computer Science* 2010

[2] Shri Prakash Dwivedi, An Efficient Multiplication Algorithm Using Nikhilam Method, *arXiv:1307.2735*, 2013

[3] David Harvey, Joris van der Hoeven and Grégoire Lecerf, Even faster integer multiplication, *arXiv:1407.3360*, 2014

[4] Anindya De, Piyush P Kurur, Chandan Saha and Ramprasad Saptharishi, Fast Integer Multiplication Using Modular Arithmetic, *Symposium on Theory of Computing* 2008

[5] Martin Fürer, Faster Integer Multiplication, *Symposium on Theory of Computing*, 2007

[6] A. A. Karatsuba, The Complexity of Computations, *Steklov Institute of Mathematics*, 1995