# Fast Integer Multiplication Using Modular Arithmetic * 

Anindya $\mathrm{De}^{\dagger}$<br>Computer Science Division<br>University of California at Berkeley<br>Berkeley, CA 94720, USA<br>anindya@eecs.berkeley.edu<br>Chandan Saha<br>Dept. of Computer Science and Engineering<br>Indian Institute of Technology Kanpur<br>Kanpur, UP, India, 208016<br>csaha@cse.iitk.ac.in

Piyush P Kurur ${ }^{\ddagger}$<br>Dept. of Computer Science and Engineering<br>Indian Institute of Technology Kanpur<br>Kanpur, UP, India, 208016<br>ppk@cse.iitk.ac.in

Ramprasad Saptharishi ${ }^{\S}$
Chennai Mathematical Institute
Plot H1, SIPCOT IT Park
Padur PO, Siruseri, India, 603103
ramprasad@cmi.ac.in


#### Abstract

We give an $N \cdot \log N \cdot 2^{O\left(\log ^{*} N\right)}$ time algorithm to multiply two $N$-bit integers that uses modular arithmetic for intermediate computations instead of arithmetic over complex numbers as in Fürer's algorithm, which also has the same and so far the best known complexity. The previous best algorithm using modular arithmetic (by Schönhage and Strassen) has complexity $O(N \cdot \log N \cdot \log \log N)$. The advantage of using modular arithmetic as opposed to complex number arithmetic is that we can completely evade the task of bounding the truncation error due to finite approximations of complex numbers, which makes the analysis relatively simple. Our algorithm is based upon Fürer's algorithm, but uses FFT over multivariate polynomials along with an estimate of the least prime in an arithmetic progression to achieve this improvement in the modular setting. It can also be viewed as a $p$-adic version of Fürer's algorithm.


## 1 Introduction

Computing the product of two $N$-bit integers is nearly a ubiquitous operation in algorithm design. Being a basic arithmetic operation, it is no surprise that multiplications of integers occur as intermediate steps of computation in algorithms from every possible domain of computer science. But seldom do the complexity of such multiplications influence the overall efficiency of the algorithm as the integers involved are relatively small in size and the multiplications can often be implemented as fast hardware operations. However, with the advent of modern cryptosystems, the study of the bit complexity of integer multiplication received a significant impetus. Indeed, large integer multiplication forms the foundation of many modern day public-key crystosytems, like RSA, El-Gamal and Elliptic Curve crytosystems. One of the most notable applications is the RSA cryptosystem,

[^0]where it is required to multiply two primes that are hundreds or thousands of bits long. The larger these primes the harder it is to factor their product, which in turn makes the RSA extremely secure in practice.

In this paper, our focus is more on the theoretical aspects of integer multiplication, it being a fundamental problem in its own right. This is to say, we will be concerned with the asymptotic bit complexity of mutiplying two $N$-bit integers with little emphasis on optimality in practice. We begin with a brief account of earlier work on integer multiplication algorithms.

### 1.1 Previous Work

The naive approach to multiply two $N$-bit integers leads to an algorithm that uses $O\left(N^{2}\right)$ bit operations. Karatsuba KO63 showed that some multiplication operations of such an algorithm can be replaced by less costly addition operations which reduces the overall running time of the algorithm to $O\left(N^{\log _{2} 3}\right)$ bit operations. Shortly afterwards, this result was improved by Toom Too63 who showed that for any $\varepsilon>0$, integer multiplication can be done in $O\left(N^{1+\varepsilon}\right)$ time. This led to the question as to whether the time complexity can be improved further by replacing the term $O\left(N^{\epsilon}\right)$ by a poly-logarithmic factor. In a major breakthrough, Schönhage and Strassen [SS71] gave two efficient algorithms for multiplying integers using fast polynomial multiplication. One of the algorithms achieved a running time of $O\left(N \cdot \log N \cdot \log \log N \ldots 2^{O\left(\log ^{*} N\right)}\right)$ using arithmetic over complex numbers (approximated to suitable precisions), while the other used arithmetic modulo carefully chosen integers to improve the complexity further to $O(N \cdot \log N \cdot \log \log N)$ bit operations. The modular algorithm remained the best for a long period of time until a recent remarkable result by Fürer [Für07] (see also [Für09]). Fürer gave an algorithm that uses arithmetic over complex numbers and runs in $N \cdot \log N \cdot 2^{O\left(\log ^{*} N\right)}$ time. Till date this is the best time complexity known for integer multiplication and indeed our result is inspired by Fürer's algorithm.

Further details on other approaches and enhancements to previous integer multiplication algorithms can be found in Für09.

### 1.2 The Motivation

Schönhage and Strassen introduced two seemingly different approaches to integer multiplication - using complex and modular arithmetic. Fürer's algorithm improves the time complexity in the complex arithmetic setting by cleverly reducing some costly multiplications to simple shift operations. However, the algorithm needs to approximate the complex numbers to certain precisions during computation. This introduces the added task of bounding the total truncation errors in the analysis of the algorithm. On the contrary, in the modular setting the error analysis is virtually absent or rather more implicit, which in turn simplifies the overall analysis. In addition, modular arithmetic gives a discrete approach to a discrete problem like integer multiplication. Therefore, it seems natural to ask whether we can achieve a similar improvement in time complexity of this problem in the modular arithmetic setting. In this work, we answer this question affirmatively. We give an $N \cdot \log N \cdot 2^{O\left(\log ^{*} N\right)}$ time algorithm for integer multiplication using only modular arithmetic, thus matching the improvement made by Fürer.

## Overview of our result

As is the case in both Schönhage-Strassen's and Fürer's algorithms, we start by reducing the problem to polynomial multiplication over a ring $\mathcal{R}$ by properly encoding the given integers. Polynomials can be multiplied efficiently using Discrete Fourier Transforms (DFT). However, in order that we are able to use Fast Fourier Transform (FFT), the ring $\mathcal{R}$ should have some special roots of unity. For instance, to multiply two polynomials of degree less than $M$ using FFT, we require a principal $2 M$-th root of unity (see Definition 2.1 for principal roots). One way to construct such a ring in the modular setting is to consider rings of the form $\mathcal{R}=\mathbb{Z} /\left(2^{M}+1\right) \mathbb{Z}$ as in Schönhage and Strassen's work SS71. In this case, the element 2 is a $2 M$-th principal root of unity in $\mathcal{R}$. This approach can be equivalently viewed as attaching an 'artificial' root to the ring of integers. However, this makes the size of $\mathcal{R}$ equal to $2^{M}$ and thus a representation of an arbitrary element in $\mathcal{R}$ takes $M$ bits. This means an $N$-bit integer is encoded as a polynomial of degree $M$ with every coefficient about $M$ bits long, thereby making $M \approx \sqrt{N}$ as the optimal choice. Indeed, the choice of such an $\mathcal{R}$ is the basis of Schönhage and Strassen's modular algorithm in which they reduce multiplication of $N$-bit integers to multiplication of $\sqrt{N}$-bit integers and achieve a complexity of $O(N \cdot \log N \cdot \log \log N)$ bit operations.

Naturally, such rings are a little too expensive in our setting. We would rather like to find a ring whose size is bounded by some polynomial in $M$ and which still contains a principal $2 M$-th root of unity. In fact, it is this task of choosing a suitable ring that poses the primary challenge in adapting Fürer's algorithm and making it work in the discrete setting.

We choose the ring to be $\mathcal{R}=\mathbb{Z} / p^{c} \mathbb{Z}$, for a prime $p$ and a constant $c$ such that $p^{c}=\operatorname{poly}(M)$. The ring $\mathbb{Z} / p^{c} \mathbb{Z}$, has a principal $2 M$-th root of unity if and only if $2 M$ divides $p-1$, which means that we need to find a prime $p$ from the arithmetic progression $\{1+i \cdot 2 M\}_{i>0}$. To make this search computationally efficient, we also need the degree of the polynomials, $M$ to be sufficiently small. This we can achieve by encoding the integers as multivariate polynomials instead of univariate ones. It turns out that the choice of the ring as $\mathcal{R}=\mathbb{Z} / p^{c} \mathbb{Z}$ is still not quite sufficient and needs a little more refinement. This is explained in Section 2.1.

The use of multivariate polynomial multiplications along with a small base ring are the main steps where our algorithm differs from earlier algorithms by Schönhage-Strassen and Fürer. Towards understanding the notion of inner and outer DFT in the context of multivariate polynomials, we also present a group theoretic interpretation of Discrete Fourier Transform (DFT). The use of inner and outer DFT plays a central role in both Fürer's as well as our algorithm. Arguing along the line of Fürer [Für07] we show that repeated use of efficient computation of inner DFT's using some special roots of unity in $\mathcal{R}$ makes the overall process efficient and leads to an $N \cdot \log N \cdot 2^{O\left(\log ^{*} N\right)}$ time algorithm.

## 2 The Basic Setup

### 2.1 The Underlying Ring

Rings of the form $\mathcal{R}=\mathbb{Z} /\left(2^{M}+1\right) \mathbb{Z}$ have the nice property that multiplications by powers of 2 , the $2 M$-th principal root of unity, are mere shift operations and are therefore very efficient. Although by choosing the ring $\mathcal{R}=\mathbb{Z} / p^{c} \mathbb{Z}$ we ensure that the ring size is small, it comes with a price: multiplications by principal roots of unity are no longer just shift operations. Fortunately, this can be redeemed by working with rings of the form $\mathcal{R}=\mathbb{Z}[\alpha] /\left(p^{c}, \alpha^{m}+1\right)$ for some $m$ whose value will
be made precise later. Elements of $\mathcal{R}$ are thus $m-1$ degree polynomials over $\alpha$ with coefficients from $\mathbb{Z} / p^{c} \mathbb{Z}$. By construction, $\alpha$ is a $2 m$-th root of unity and multiplication of any element in $\mathcal{R}$ by any power of $\alpha$ can be achieved by shift operations - this property is crucial in making some multiplications in the FFT less costly (see Section 3.2).

Given an $N$-bit number $a$, we encode it as a $k$-variate polynomial over $\mathcal{R}$ with degree in each variable less than $M$. The parameters $M$ and $m$ are powers of two such that $M^{k}$ is roughly $\frac{N}{\log ^{2} N}$ and $m$ is roughly $\log N$. The parameter $k$ will ultimately be chosen a constant (see Section 4.2). We now explain the details of this encoding process.

### 2.2 Encoding Integers into $k$-variate Polynomials

Given an $N$-bit integer $a$, we first break these $N$ bits into $M^{k}$ blocks of roughly $\frac{N}{M^{k}}$ bits each. This corresponds to representing $a$ in base $q=2^{\frac{N}{M^{k}}}$. Let $a=a_{0}+\ldots+a_{M^{k}-1} q^{M^{k}-1}$, where every $a_{i}<q$. The number $a$ is converted into a polynomial as follows:

1. Express $i$ in base $M$ as $i=i_{1}+i_{2} M+\cdots+i_{k} M^{k-1}$.
2. Encode each term $a_{i} q^{i}$ as the monomial $a_{i} \cdot X_{1}^{i_{1}} X_{2}^{i_{1}} \cdots X_{k}^{i_{k}}$. As a result, the number $a$ gets converted to the polynomial $\sum_{i=0}^{M^{k}-1} a_{i} \cdot X_{1}^{i_{1}} \cdots X_{k}^{i_{k}}$.

Further, we break each $a_{i}$ into $\frac{m}{2}$ equal sized blocks where the number of bits in each block is $u=\frac{2 N}{M^{k} \cdot m}$. Each coefficient $a_{i}$ is then encoded as a polynomial in $\alpha$ of degree less than $\frac{m}{2}$. The polynomials are then padded with zeroes to stretch their degrees to $m$. Thus, the $N$-bit number $a$ is converted to a $k$-variate polynomial $a(X)$ over $\mathbb{Z}[\alpha] /\left(\alpha^{m}+1\right)$.

Given integers $a$ and $b$, each of $N$ bits, we encode them as polynomials $a(X)$ and $b(X)$ and compute the product polynomial. The product $a \cdot b$ can be recovered by substituting $X_{s}=q^{M^{s-1}}$, for $1 \leq s \leq k$, and $\alpha=2^{u}$ in the polynomial $a(X) \cdot b(X)$. The coefficients in the product polynomial could be as large as $M^{k} \cdot m \cdot 2^{2 u}$ and hence it is sufficient to do arithmetic modulo $p^{c}$ where $p^{c}>M^{k} \cdot m \cdot 2^{2 u}$. Our choice of the prime $p$ ensures that $c$ is in fact a constant (see Section 4.2).

### 2.3 Choosing the Prime

The prime $p$ should be chosen such that the ring $\mathbb{Z} / p^{c} \mathbb{Z}$ has a principal $2 M$-th root of unity, which is required for polynomial multiplication using FFT. A principal root of unity is defined as follows.

Definition 2.1. (Principal root of unity) An n-th root of unity $\zeta \in \mathcal{R}$ is said to be primitive if it generates a cyclic group of order $n$ under multiplication. Furthermore, it is said to be principal if $n$ is coprime to the characteristic of $\mathcal{R}$ and $\zeta$ satisfies $\sum_{i=0}^{n-1} \zeta^{i j}=0$ for all $0<j<n$.

In $\mathbb{Z} / p^{c} \mathbb{Z}$, a $2 M$-th root of unity is principal if and only if $2 M \mid p-1$ (see also Section 5). As a result, we need to choose the prime $p$ from the arithmetic progression $\{1+i \cdot 2 M\}_{i>0}$, which is potentially the main bottleneck of our approach. We now explain how to circumvent this problem.

An upper bound for the least prime in an arithmetic progression is given by the following theorem by Linnik Lin44:

Theorem 2.2. (Linnik) There exist absolute constants $\ell$ and $L$ such that for any pair of coprime integers $d$ and $n$, the least prime $p$ such that $p \equiv d \bmod n$ is less than $\ell n^{L}$.

Heath-Brown HB92 showed that the Linnik constant $L \leq 5.5$ (a recent work by Xylouris $\mathrm{Xyl10}$ showed that $L \leq 5.2$ ). Recall that $M$ is chosen such that $M^{k}$ is $O\left(\frac{N}{\log ^{2} N}\right)$. If we choose $k=1$, that is if we use univariate polynomials to encode integers, then the parameter $M=$ $O\left(\frac{N}{\log ^{2} N}\right)$. Hence the least prime $p \equiv 1(\bmod 2 M)$ could be as large as $N^{L}$. Since all known deterministic sieving procedures take at least $N^{L}$ time this is clearly infeasible (for a randomized approach see Section 4.2). However, by choosing a larger $k$ we can ensure that the least prime $p \equiv 1(\bmod 2 M)$ is $O\left(N^{\varepsilon}\right)$ for some constant $\varepsilon<1$.

Remark 2.3. If $k$ is any integer greater than $L+1$, then $M^{L}=O\left(N^{\frac{L}{L+1}}\right)$ and hence the least prime $p \equiv 1(\bmod 2 M)$ can be found in $o(N)$ time.

### 2.4 Finding the Root of Unity

We require a principal $2 M$-th root of unity $\rho(\alpha)$ in $\mathcal{R}$ to compute the Fourier transforms. This root $\rho(\alpha)$ should also have the property that its $\left(\frac{M}{m}\right)$-th power is $\alpha$, so as to make some multiplications in the FFT efficient (see Section 3.2 ). The root $\rho(\alpha)$ can be computed by interpolation in a way similar to that in Fürer's algorithm [Für07, Section 3], except that we need a principal $2 M$-th root of unity $\omega$ in $\mathbb{Z} / p^{c} \mathbb{Z}$ to start with. To obtain such a root, we first obtain a $(p-1)$-th root of unity $\zeta$ in $\mathbb{Z} / p^{c} \mathbb{Z}$ by lifting a generator of $\mathbb{F}_{p}^{\times}$. The $\left(\frac{p-1}{2 M}\right)$-th power of $\zeta$ gives us the required $2 M$-th root of unity $\omega$. A generator of $\mathbb{F}_{p}^{\times}$can be computed by brute force, as $p$ is sufficiently small. Having obtained a generator, we use Hensel Lifting [NZM91, Theorem 2.23].

Lemma 2.4 (Hensel Lifting). Let $\zeta_{s}$ be a primitive $(p-1)$-th root of unity in $\mathbb{Z} / p^{s} \mathbb{Z}$. Then there exists a unique primitive $(p-1)$-th root of unity $\zeta_{s+1}$ in $\mathbb{Z} / p^{s+1} \mathbb{Z}$ such that $\zeta_{s+1} \equiv \zeta_{s}\left(\bmod p^{s}\right)$. This unique root is given by $\zeta_{s+1}=\zeta_{s}-\frac{f\left(\zeta_{s}\right)}{f^{\prime}\left(\zeta_{s}\right)}$ where $f(x)=x^{p-1}-1$.

It can be shown that the root $\zeta$ in $\mathbb{Z} / p^{c} \mathbb{Z}$ thus obtained by lifting is principal. Furthermore, different powers of $\zeta$ are distinct modulo $p$. Therefore, the difference between any two of them is a unit in $\mathbb{Z} / p^{c} \mathbb{Z}$ and this makes the following interpolation feasible in our setting.

Finding $\rho(\alpha)$ from $\omega$ : Since $\omega$ is a principal $2 M$-th root of unity, $\gamma=\omega^{\frac{2 M}{2 m}}$ is a principal $2 m$-th root of unity in $\mathbb{Z} / p^{c} \mathbb{Z}$. Notice that, $\alpha^{m}+1$ uniquely factorizes as, $\alpha^{m}+1=(\alpha-\gamma)\left(\alpha-\gamma^{3}\right) \ldots(\alpha-$ $\gamma^{2 m-1}$ ), and the ideals generated by $\left(\alpha-\gamma^{i}\right)$ in $\mathcal{R}$ are mutually coprime as $\gamma^{i}-\gamma^{j}$ is a unit for $i \neq j$. Therefore, using Chinese Remaindering, $\alpha$ has the direct sum representation $\left(\gamma, \gamma^{3}, \ldots, \gamma^{2 m-1}\right)$ in $\mathcal{R}$. Since we require $\rho(\alpha)^{\frac{2 M}{2 m}}=\alpha$, it is sufficient to choose a $\rho(\alpha)$ whose direct sum representation is $\left(\omega, \omega^{3}, \ldots, \omega^{2 m-1}\right)$. Now use Lagrange's formula to interpolate $\rho(\alpha)$ as,

$$
\rho(\alpha)=\sum_{i=1, i \text { odd }}^{2 m-1} \omega^{i} \cdot \prod_{j=1,}^{2 m-1} \frac{\alpha-\gamma^{j}}{\gamma^{i}-\gamma^{j}}
$$

The inverses of the elements $\gamma^{i}-\gamma^{j}$ can be easily computed in $\mathbb{Z} / p^{c} \mathbb{Z}$.

## 3 Fourier Transform

### 3.1 Inner and Outer DFT

Suppose that $a(x) \in \mathcal{S}[x]$ is a polynomial of degree less than $M$, where $\mathcal{S}$ is a ring containing a $2 M$ th principal root of unity $\rho$. Let us say that we want to compute the $2 M$-point DFT of $a(x)$ using $\rho$ as the root of unity. In other words, we want to compute the elements $a(1), a(\rho), \ldots, a\left(\rho^{2 M-1}\right)$ is $\mathcal{S}$. This can be done in two steps.

Step 1: Compute the following polynomials using $\alpha=\rho^{2 M / 2 m}$.

$$
\begin{aligned}
a_{0}(x) & =a(x) \bmod \left(x^{2 M / 2 m}-1\right) \\
a_{1}(x) & =a(x) \bmod \left(x^{2 M / 2 m}-\alpha\right) \\
& \vdots \\
a_{2 m-1}(x) & =a(x) \bmod \left(x^{2 M / 2 m}-\alpha^{2 m-1}\right)
\end{aligned}
$$

where $\operatorname{deg}\left(a_{j}(x)\right)<\frac{2 M}{2 m}$ for all $0 \leq j<2 m$.
Step 2: Note that, $a_{j}\left(\rho^{k \cdot 2 m+j}\right)=a\left(\rho^{k \cdot 2 m+j}\right)$ for every $0 \leq j<2 m$ and $0 \leq k<\frac{2 M}{2 m}$. Therefore, all we need to do to compute the DFT of $a(x)$ is to evaluate the polynomials $a_{j}(x)$ at appropriate powers of $\rho$.

The idea is to show that both Step 1 and Step 2 can be performed by computation of some 'smaller' DFTs. Let us see how.

Performing Step 1: The crucial observation here is the following. Fix an integer $\ell$ in the range $\left[0, \frac{2 M}{2 m}-1\right]$. Then the $\ell^{t h}$ coefficients of $a_{0}(x), a_{1}(x), \ldots, a_{2 m-1}(x)$ are exactly $e_{\ell}(1), e_{\ell}(\alpha), \ldots, e_{\ell}\left(\alpha^{2 m-1}\right)$, respectively, where $e_{\ell}(y)$ is the polynomial,

$$
e_{\ell}(y)=\sum_{j=0}^{2 m-1} a_{j \cdot \frac{2 M}{2 m}+\ell} \cdot y^{j}
$$

But then, finding $e_{\ell}(1), e_{\ell}(\alpha), \ldots, e_{\ell}\left(\alpha^{2 m-1}\right)$ is essentially computing the $2 m$-point DFT of $e_{\ell}(y)$ using $\alpha$ as the $2 m^{t h}$ root of unity. Therefore, all we need to do to find $a_{0}(x), \ldots, a_{2 m-1}(x)$ is to compute the DFTs of $e_{\ell}(y)$ for all $0 \leq \ell<\frac{2 M}{2 m}$. These $\frac{2 M}{2 m}$ many $2 m$-point DFTs are called the inner DFTs.

Performing Step 2: In order to find $a_{j}\left(\rho^{k \cdot 2 m+j}\right)$, for $0 \leq k<\frac{2 M}{2 m}$ and a fixed $j$, we first compute the polynomial $\tilde{a}_{j}(x)=a_{j}\left(x \cdot \rho^{j}\right)$ followed by a $\frac{2 M}{2 m}$-point DFT of $\tilde{a}_{j}(x)$ using $\rho^{2 m}$ as the root of unity. These $2 m$ many $\frac{2 M}{2 m}$-point DFTs ( $j$ running from 0 to $2 m-1$ ) are called the outer DFTs. The polynomials $\tilde{a}_{j}(x)$ can be computed by multiplying the coefficients of $a_{j}(x)$ by suitable powers of $\rho$. Such multiplications are termed as bad multiplications.

The above discussion is summarized in the following lemma.

Lemma 3.1. $($ DFT time $=$ Inner DFTs + Bad multiplications + Outer DFTs)
Time taken to compute a $2 M$-point DFT over $\mathcal{S}$ is sum of:

1. Time taken to compute $\frac{2 M}{2 m}$ many $2 m$-point inner DFTs over $\mathcal{S}$ using $\alpha$ as the $2 m$-th root of unity.
2. Time to do $2 M$ multiplications in $\mathcal{S}$ by powers of $\rho$ (bad multiplications).
3. Time taken to compute $2 m$ many $\frac{2 M}{2 m}$-point outer DFTs over $\mathcal{S}$ using $\rho^{2 m}$ as the $\frac{2 M}{2 m}$-th root of unity.

### 3.2 Analysis of the FFT

We are now ready to analyse the complexity of multiplying the two $k$-variate polynomials $a(X)$ and $b(X)$ (see Section 2.2) using Fast Fourier Transform. Treat $a(X)$ and $b(X)$ as univariate polynomials in variable $X_{k}$ over the ring $\mathcal{S}=\mathcal{R}\left[X_{1}, \ldots, X_{k-1}\right]$. We write $a(X)$ and $b(X)$ as $a\left(X_{k}\right)$ and $b\left(X_{k}\right)$, respectively, where $\operatorname{deg}\left(a\left(X_{k}\right)\right)$ and $\operatorname{deg}\left(b\left(X_{k}\right)\right)$ are less than $M$. Multiplication of $a(X)$ and $b(X)$ can be thought of as multiplication of the univariates $a\left(X_{k}\right)$ and $b\left(X_{k}\right)$ over $\mathcal{S}$. Also note that, the root $\rho(\alpha)$ (constructed in Section 2.4) is a primitive $2 M$-th root of unity in $\mathcal{S} \supset \mathcal{R}$. Denote the multiplication complexity of $a\left(X_{k}\right)$ and $b\left(X_{k}\right)$ by $\mathcal{F}(2 M, k)$.

Multiplication of $a\left(X_{k}\right)$ and $b\left(X_{k}\right)$ using FFT involves computation of three $2 M$-point DFTs over $\mathcal{S}$ and $2 M$ pointwise (or componentwise) multiplications in $\mathcal{S}$. Let $\mathcal{D}(2 M, k)$ be the time taken to compute a $2 M$-point DFT over $\mathcal{S}$. By Lemma 3.1, the time to compute a DFT is the sum of the time for the inner DFTs, the bad multiplications and the outer DFTs. Let us analyse these three terms separately. We will go by the notation in Section 3.1, using $\mathcal{S}=\mathcal{R}\left[X_{1}, \ldots, X_{k-1}\right]$ and $\rho=\rho(\alpha)$.

Inner DFT time: Computing a $2 m$-point DFT requires $2 m \log (2 m)$ additions in $\mathcal{S}$ and $m \log (2 m)$ multiplications by powers of $\alpha$. The important observation here is: since $\mathcal{R}=\mathbb{Z}[\alpha] /\left(p^{c}, \alpha^{m}+1\right)$, multiplication by a power of $\alpha$ with an element in $\mathcal{R}$ can be readily computed by simple cyclic shifts (with possible negations), which takes only $O(m \cdot \log p)$ bit operations. An element in $\mathcal{S}$ is just a polynomial over $\mathcal{R}$ in variables $X_{1}, \ldots, X_{k-1}$, with degree in each variable bounded by $M$. Hence, multiplication by a power of $\alpha$ with an element of $\mathcal{S}$ can be done using $\mathcal{N}_{\mathcal{S}}=O\left(M^{k-1} \cdot m \cdot \log p\right)$ bit operations. A total of $m \log (2 m)$ multiplications takes $O\left(m \log m \cdot \mathcal{N}_{\mathcal{S}}\right)$ bit operations. It is easy to see that $2 m \log (2 m)$ additions in $\mathcal{S}$ also require the same order of time.

Since there are $\frac{2 M}{2 m}$ many $2 m$-point DFTs, the total time spent in the inner DFTs is $O(2 M$. $\left.\log m \cdot \mathcal{N}_{\mathcal{S}}\right)$ bit operations.

Bad multiplication time: Suppose that two arbitrary elements in $\mathcal{R}$ can be multiplied using $\mathcal{M}_{\mathcal{R}}$ bit operations. Mulitplication in $\mathcal{S}$ by a power of $\rho$ amounts to $c_{\mathcal{S}}=M^{k-1}$ multiplications in $\mathcal{R}$. Since there are $2 M$ such bad multiplications, the total time is bounded by $O\left(2 M \cdot c_{\mathcal{S}} \cdot \mathcal{M}_{\mathcal{R}}\right)$.

Outer DFT time: By Lemma 3.1, the total outer DFT time is $2 m \cdot \mathcal{D}\left(\frac{2 M}{2 m}, k\right)$.

Total DFT time: Therefore, the net DFT time is bounded as,

$$
\begin{aligned}
\mathcal{D}(2 M, k) & =O\left(2 M \cdot \log m \cdot \mathcal{N}_{\mathcal{S}}+2 M \cdot c_{\mathcal{S}} \cdot \mathcal{M}_{\mathcal{R}}\right)+2 m \cdot \mathcal{D}\left(\frac{2 M}{2 m}, k\right) \\
& =O\left(2 M \cdot \log m \cdot \mathcal{N}_{\mathcal{S}}+2 M \cdot c_{\mathcal{S}} \cdot \mathcal{M}_{\mathcal{R}}\right) \cdot \frac{\log 2 M}{\log 2 m} \\
& =O\left(M^{k} \log M \cdot m \log p+\frac{M^{k} \log M}{\log m} \cdot \mathcal{M}_{\mathcal{R}}\right)
\end{aligned}
$$

putting the values of $\mathcal{N}_{\mathcal{S}}$ and $c_{\mathcal{S}}$.
Pointwise multiplications: Finally, FFT does $2 M$ pointwise multiplications in $\mathcal{S}$. Since elements of $\mathcal{S}$ are $(k-1)$-variate polynomials over $\mathcal{R}$, with degree in every variable bounded by $M$, the total time taken for pointwise multiplications is $2 M \cdot \mathcal{F}(2 M, k-1)$ bit operations.

Total polynomial multiplication time: This can be expressed as,

$$
\begin{align*}
\mathcal{F}(2 M, k) & =O\left(M^{k} \log M \cdot m \log p+\frac{M^{k} \log M}{\log m} \cdot \mathcal{M}_{\mathcal{R}}\right)+2 M \cdot \mathcal{F}(2 M, k-1) \\
& =O\left(M^{k} \log M \cdot m \log p+\frac{M^{k} \log M}{\log m} \cdot \mathcal{M}_{\mathcal{R}}\right) \tag{1}
\end{align*}
$$

as $k$ is a constant.

We now present an equivalent group theoretic interpretation of the above process of polynomial multiplication, which is a subject of interest in itself.

### 3.3 A Group Theoretic Interpretation

A convenient way to study polynomial multiplication is to interpret it as multiplication in a group algebra.

Definition 3.2. (Group Algebra) Let $G$ be any group. The group algebra of $G$ over a ring $R$ is the set of formal sums $\sum_{g \in G} \alpha_{g} g$ where $\alpha_{g} \in R$ with addition defined point-wise and multiplication defined via convolution as follows

$$
\left(\sum_{g} \alpha_{g} g\right)\left(\sum_{h} \beta_{h} h\right)=\sum_{u}\left(\sum_{g h=u} \alpha_{g} \beta_{h}\right) u
$$

Multiplying univariate polynomials over $R$ of degree less than $n$ can be seen as multiplication in the group algebra $R[G]$ where $G$ is the cyclic group of order $2 n$. Similarly, multiplying $k$-variate polynomials of degree less than $n$ in each variable can be seen as multiplying in the group algebra $R\left[G^{k}\right]$, where $G^{k}$ denotes the $k$-fold product group $G \times \ldots \times G$.

In this section, we study the Fourier transform over the group algebra $R[E]$ where $E$ is an additive abelian group. Most of this, albeit in a different form, is well known but is provided here for completeness [Sha99, Chapter 17].

In order to simplify our presentation, we will fix the base ring to be $\mathbb{C}$, the field of complex numbers. Let $n$ be the exponent of $E$, that is the maximum order of any element in $E$. A similar approach can be followed for any other base ring as long as it has a principal $n$-th root of unity.

We consider $\mathbb{C}[E]$ as a vector space with basis $\{x\}_{x \in E}$ and use the Dirac notation to represent elements of $\mathbb{C}[E]$ - the vector $|x\rangle, x$ in $E$, denotes the element 1.x of $\mathbb{C}[E]$.

Definition 3.3. (Characters) Let $E$ be an additive abelian group. A character of $E$ is a homomorphism from $E$ to $\mathbb{C}^{*}$.

An example of a character of $E$ is the trivial character, which we will denote by 1 , that assigns to every element of $E$ the complex number 1 . If $\chi_{1}$ and $\chi_{2}$ are two characters of $E$ then their product $\chi_{1} \cdot \chi_{2}$ is defined as $\chi_{1} \cdot \chi_{2}(x)=\chi_{1}(x) \chi_{2}(x)$.

Proposition 3.4. Sha99, Chapter 17, Theorem 1] Let E be an additive abelian group of exponent $n$. Then the values taken by any character of $E$ are n-th roots of unity. Furthermore, the characters form a multiplicative abelian group $\hat{E}$ which is isomorphic to $E$.

An important property that the characters satisfy is the following [Isa94, Corollary 2.14].
Proposition 3.5. (Schur's Orthogonality) Let $E$ be an additive abelian group. Then

$$
\sum_{x \in E} \chi(x)=\left\{\begin{array}{ll}
0 & \text { if } \chi \neq 1, \\
\# E & \text { otherwise }
\end{array} \quad \text { and } \quad \sum_{\chi \in \hat{E}} \chi(x)= \begin{cases}0 & \text { if } x \neq 0 \\
\# E & \text { otherwise }\end{cases}\right.
$$

It follows from Schur's orthogonality that the collection of vectors $|\chi\rangle=\sum_{x} \chi(x)|x\rangle$ forms a basis of $\mathbb{C}[E]$. We will call this basis the Fourier basis of $\mathbb{C}[E]$.

Definition 3.6. (Fourier Transform) Let $E$ be an additive abelian group and let $x \mapsto \chi_{x}$ be an isomorphism between $E$ and $\hat{E}$. The Fourier transform over $E$ is the linear map from $\mathbb{C}[E]$ to $\mathbb{C}[E]$ that sends $|x\rangle$ to $\left|\chi_{x}\right\rangle$.

Thus, the Fourier transform is a change of basis from the point basis $\{|x\rangle\}_{x \in E}$ to the Fourier basis $\left\{\left|\chi_{x}\right\rangle\right\}_{x \in E}$. The Fourier transform is unique only up to the choice of the isomorphism $x \mapsto \chi_{x}$. This isomorphism is determined by the choice of the principal root of unity.

Remark 3.7. Given an element $|f\rangle \in \mathbb{C}[E]$, to compute its Fourier transform it is sufficient to compute the Fourier coefficients $\{\langle\chi \mid f\rangle\}_{\chi \in \hat{E}}$.

## Fast Fourier Transform

We now describe the Fast Fourier Transform for general abelian groups in the character theoretic setting. For the rest of the section fix an additive abelian group $E$ over which we would like to compute the Fourier transform. Let $A$ be any subgroup of $E$ and let $B=E / A$. For any such pair of abelian groups $A$ and $B$, we have an appropriate Fast Fourier transformation, which we describe in the rest of the section.

Proposition 3.8. 1. Every character $\lambda$ of $B$ can be "lifted" to a character of $E$ (which will also be denoted by $\lambda$ ) defined as follows $\lambda(x)=\lambda(x+A)$.
2. Let $\chi_{1}$ and $\chi_{2}$ be two characters of $E$ that when restricted to $A$ are identical. Then $\chi_{1}=\chi_{2} \lambda$ for some character $\lambda$ of $B$.
3. The group $\hat{B}$ is (isomorphic to) a subgroup of $\hat{E}$ with the quotient group $\hat{E} / \hat{B}$ being (isomorphic to) $\hat{A}$.

We now consider the task of computing the Fourier transform of an element $|f\rangle=\sum f_{x}|x\rangle$ presented as a list of coefficients $\left\{f_{x}\right\}$ in the point basis. For this, it is sufficient to compute the Fourier coefficients $\{\langle\chi \mid f\rangle\}$ for each character $\chi$ of $E$ (Remark 3.7). To describe the Fast Fourier transform we fix two sets of cosets representatives, one of $A$ in $E$ and one of $\hat{B}$ in $\hat{E}$ as follows.

1. For each $b \in B, b$ being a coset of $A$, fix a coset representative $x_{b} \in E$ such $b=x_{b}+A$.
2. For each character $\varphi$ of $A$, fix a character $\chi_{\varphi}$ of $E$ such that $\chi_{\varphi}$ restricted to $A$ is the character $\varphi$. The characters $\left\{\chi_{\varphi}\right\}$ form (can be thought of as) a set of coset representatives of $\hat{B}$ in $\hat{E}$.

Since $\left\{x_{b}\right\}_{b \in B}$ forms a set of coset representatives, any $|f\rangle \in \mathbb{C}[E]$ can be written uniquely as $|f\rangle=\sum f_{b, a}\left|x_{b}+a\right\rangle$.

Proposition 3.9. Let $|f\rangle=\sum f_{b, a}\left|x_{b}+a\right\rangle$ be an element of $\mathbb{C}[E]$. For each $b \in B$ and $\varphi \in \hat{A}$ let $\left|f_{b}\right\rangle \in \mathbb{C}[A]$ and $\left|f_{\varphi}\right\rangle \in \mathbb{C}[B]$ be defined as follows.

$$
\begin{aligned}
\left|f_{b}\right\rangle & =\sum_{a \in A} f_{b, a}|a\rangle \\
\left|f_{\varphi}\right\rangle & =\sum_{b \in B} \bar{\chi}_{\varphi}\left(x_{b}\right)\left\langle\varphi \mid f_{b}\right\rangle|b\rangle
\end{aligned}
$$

Then for any character $\chi=\chi_{\varphi} \lambda$ of $E$ the Fourier coefficient $\langle\chi \mid f\rangle=\left\langle\lambda \mid f_{\varphi}\right\rangle$.
Proof.

$$
\langle\chi \mid f\rangle=\sum_{b \in B, a \in A} \overline{\chi_{\varphi} \lambda\left(x_{b}+a\right)} \cdot f_{b, a}
$$

Recall that for any $\lambda$, that is a lift of a character of $B$, acts identically inside cosets of $A$. Therefore, the above sum can be rewritten as follows:

$$
\begin{aligned}
\sum_{b \in B, a \in A} \overline{\chi_{\varphi} \lambda\left(x_{b}+a\right)} \cdot f_{b, a} & =\sum_{b \in B} \sum_{a \in A} \overline{\chi_{\varphi}\left(x_{b}+a\right) \lambda\left(x_{b}+a\right)} \cdot f_{b, a} \\
& =\sum_{b \in B} \overline{\lambda\left(x_{b}\right)} \cdot \overline{\chi_{\varphi}\left(x_{b}\right)} \sum_{a \in A} \overline{\varphi(a)} f_{b, a}
\end{aligned}
$$

The inner sum over $a$ is precisely $\left\langle\varphi \mid f_{b}\right\rangle$ and therefore we have:

$$
\langle\chi \mid f\rangle=\sum_{b \in B} \overline{\lambda\left(x_{b}\right)} \cdot \overline{\chi_{\varphi}\left(x_{b}\right)}\left\langle\varphi \mid f_{b}\right\rangle
$$

which can be rewritten as $\left\langle\lambda \mid f_{\varphi}\right\rangle$ as claimed.
We are now ready to describe the Fast Fourier transform given an element $|f\rangle=\sum f_{x}|x\rangle$.

1. For each $b \in B$ compute the Fourier transforms of $\left|f_{b}\right\rangle$. This requires \#B many Fourier transforms over $A$.
2. As a result of the previous step we have for each $b \in B$ and $\varphi \in \hat{A}$ the Fourier coefficients $\left\langle\varphi \mid f_{b}\right\rangle$. Compute for each $\varphi$ the vectors $\left|f_{\varphi}\right\rangle=\sum_{b \in B} \bar{\chi}_{\varphi}\left(x_{b}\right)\left\langle\varphi \mid f_{b}\right\rangle|b\rangle$. This requires $\# \hat{A} . \# B=\# E$ many multiplications by roots of unity.
3. For each $\varphi \in \hat{A}$ compute the Fourier transform of $\left|f_{\varphi}\right\rangle$. This requires $\# \hat{A}=\# A$ many Fourier transforms over $B$.
4. Any character $\chi$ of $E$ is of the form $\chi_{\varphi} \lambda$ for some $\varphi \in \hat{A}$ and $\lambda \in \hat{B}$. Using Proposition 3.9 we have at the end of Step 3 all the Fourier coefficients $\langle\chi \mid f\rangle=\left\langle\lambda \mid f_{\varphi}\right\rangle$.

If the quotient group $B$ itself has a subgroup that is isomorphic to $A$ then we can apply this process recursively on $B$ to obtain a divide and conquer procedure to compute Fourier transform. In the standard FFT we use $E=\mathbb{Z} / 2^{n} \mathbb{Z}$. The subgroup $A$ is $2^{n-1} E$ which is isomorphic to $\mathbb{Z} / 2 \mathbb{Z}$ and the quotient group $B$ is $\mathbb{Z} / 2^{n-1} \mathbb{Z}$.

## Analysis of the Fourier Transform

Our goal is to multiply $k$-variate polynomials over $\mathcal{R}$, with the degree in each variable less than $M$. This can be achieved by embedding the polynomials into the algebra of the product group $E=\left(\frac{\mathbb{Z}}{2 M \cdot \mathbb{Z}}\right)^{k}$ and multiplying them as elements of the algebra. Since the exponent of $E$ is $2 M$, we require a principal $2 M$-th root of unity in the ring $\mathcal{R}$. We shall use the root $\rho(\alpha)$ (as defined in Section (2.4) for the Fourier transform over $E$.

For every subgroup $A$ of $E$, we have a corresponding FFT. We choose the subgroup $A$ as $\left(\frac{\mathbb{Z}}{2 m \cdot \mathbb{Z}}\right)^{k}$ and let $B$ be the quotient group $E / A$. The group $A$ has exponent $2 m$ and $\alpha$ is a principal $2 m$-th root of unity. Since $\alpha$ is a power of $\rho(\alpha)$, we can use it for the Fourier transform over $A$. As multiplications by powers of $\alpha$ are just shifts, this makes Fourier transform over $A$ efficient.

Let $\mathcal{F}(M, k)$ denote the complexity of computing the Fourier transform over $\left(\frac{\mathbb{Z}}{2 M \cdot \mathbb{Z}}\right)^{k}$. We have

$$
\begin{equation*}
\mathcal{F}(M, k)=\left(\frac{M}{m}\right)^{k} \mathcal{F}(m, k)+M^{k} \mathcal{M}_{\mathcal{R}}+(2 m)^{k} \mathcal{F}\left(\frac{M}{2 m}, k\right) \tag{2}
\end{equation*}
$$

where $\mathcal{M}_{\mathcal{R}}$ denotes the complexity of multiplications in $\mathcal{R}$. The first term comes from the $\# B$ many Fourier transforms over $A$ (Step 1 of FFT), the second term corresponds to the multiplications by roots of unity (Step 2) and the last term comes from the $\# A$ many Fourier transforms over $B$ (Step 3).

Since $A$ is a subgroup of $B$ as well, Fourier transforms over $B$ can be recursively computed in a similar way, with $B$ playing the role of $E$. Therefore, by simplifying the recurrence in Equation 2 we get:

$$
\begin{equation*}
\mathcal{F}(M, k)=O\left(\frac{M^{k} \log M}{m^{k} \log m} \mathcal{F}(m, k)+\frac{M^{k} \log M}{\log m} \mathcal{M}_{\mathcal{R}}\right) \tag{3}
\end{equation*}
$$

Lemma 3.10. $\mathcal{F}(m, k)=O\left(m^{k+1} \log m \cdot \log p\right)$

Proof. The FFT over a group of size $n$ is usually done by taking 2-point FFT's followed by $\frac{n}{2}$-point FFT's. This involves $O(n \log n)$ multiplications by roots of unity and additions in base ring. Using this method, Fourier transforms over $A$ can be computed with $O\left(m^{k} \log m\right)$ multiplications and additions in $\mathcal{R}$. Since each multiplication is between an element of $\mathcal{R}$ and a power of $\alpha$, this can be efficiently achieved through shifting operations. This is dominated by the addition operation, which takes $O(m \log p)$ time, since this involves adding $m$ coefficients from $\mathbb{Z} / p^{c} \mathbb{Z}$.

Therefore, from Equation 3,

$$
\mathcal{F}(M, k)=O\left(M^{k} \log M \cdot m \cdot \log p+\frac{M^{k} \log M}{\log m} \mathcal{M}_{\mathcal{R}}\right)
$$

## 4 Algorithm and Analysis

### 4.1 Integer Multiplication Algorithm

We are given two integers $a, b<2^{N}$ to multiply. We fix constants $k$ and $c$ whose values are given in Section 4.2. The algorithm is as follows:

1. Choose $M$ and $m$ as powers of two such that $M^{k} \approx \frac{N}{\log ^{2} N}$ and $m \approx \log N$. Find the least prime $p \equiv 1(\bmod 2 M)($ Remark 2.3).
2. Encode the integers $a$ and $b$ as $k$-variate polynomials $a(X)$ and $b(X)$, respectively, over the ring $\mathcal{R}=\mathbb{Z}[\alpha] /\left(p^{c}, \alpha^{m}+1\right)$ (Section 2.2).
3. Compute the root $\rho(\alpha)$ (Section 2.4).
4. Use $\rho(\alpha)$ as the principal $2 M$-th root of unity to compute the Fourier transforms of the $k$ variate polynomials $a(X)$ and $b(X)$. Multiply component-wise and take the inverse Fourier transform to obtain the product polynomial. (Sections 3.1 and 3.2)
5. Evaluate the product polynomial at appropriate powers of two to recover the integer product and return it (Section 2.2).

### 4.2 Complexity Analysis

The choice of parameters should ensure that the following constraints are satisfied:

1. $M^{k}=O\left(\frac{N}{\log ^{2} N}\right)$ and $m=O(\log N)$.
2. $M^{L}=O\left(N^{\varepsilon}\right)$, where $L$ is the Linnik constant (Theorem 2.2) and $\varepsilon$ is any constant less than 1. Recall that this makes picking the prime by brute force feasible (see Remark 2.3).
3. $p^{c}>M^{k} \cdot m \cdot 2^{2 u}$ where $u=\frac{2 N}{M^{k} m}$. This is to prevent overflows during modular arithmetic (see Section 2.2).

It is straightforward to check that $k>L+1$ and $c>5(k+1)$ satisfy the above constraints. Since $L \leq 5.2$, it is sufficient to choose $k=7$ and $c=42$.

Let $T(N)$ denote the time complexity of multiplying two $N$ bit integers. This consists of:

- Time required to pick a suitable prime $p$,
- Computing the root $\rho(\alpha)$,
- Encoding the input integers as polynomials,
- Multiplying the encoded polynomials,
- Evaluating the product polynomial.

As argued before, the prime $p$ can be chosen in $o(N)$ time. To compute $\rho(\alpha)$, we need to lift a generator of $\mathbb{F}_{p}^{\times}$to $\mathbb{Z} / p^{c} \mathbb{Z}$ followed by an interpolation. Since $c$ is a constant and $p$ is a prime of $O(\log N)$ bits, the time required for Hensel Lifting and interpolation is $o(N)$.

The encoding involves dividing bits into smaller blocks, and expressing the exponents of $q$ in base $M$ (Section 2.2) and all these take $O(N)$ time since $M$ is a power of 2 . Similarly, evaluation of the product polynomial takes linear time as well. Therefore, the time complexity is dominated by the time taken for polynomial multiplication.

## Time complexity of Polynomial Multiplication

From Equation 1, the complexity of polynomial multiplication is given by,

$$
\mathcal{F}(2 M, k)=O\left(M^{k} \log M \cdot m \cdot \log p+\frac{M^{k} \log M}{\log m} \cdot \mathcal{M}_{\mathcal{R}}\right) .
$$

Proposition 4.1. Sch82 Multiplication in $\mathcal{R}$ reduces to multiplying $O\left(\log ^{2} N\right)$ bit integers and hence $\mathcal{M}_{\mathcal{R}}=T\left(O\left(\log ^{2} N\right)\right)$.

Proof. Elements of $\mathcal{R}$ can be viewed as polynomials in $\alpha$ over $\mathbb{Z} / p^{c} \mathbb{Z}$ with degree at most $m$. Given two such polynomials $f(\alpha)$ and $g(\alpha)$, encode them as follows: Replace $\alpha$ by $2^{d}$, transforming the polynomials $f(\alpha)$ and $g(\alpha)$ to the integers $f\left(2^{d}\right)$ and $g\left(2^{d}\right)$ respectively. The parameter $d$ is chosen such that the coefficients of the product $h(\alpha)=f(\alpha) g(\alpha)$ can be recovered from the product $f\left(2^{d}\right) \cdot g\left(2^{d}\right)$. For this, it is sufficient to ensure that the maximum coefficient of $h(\alpha)$ is less than $2^{d}$. Since $f$ and $g$ are polynomials of degree $m$, we would want $2^{d}$ to be greater than $m \cdot p^{2 c}$, which can be ensured by choosing $d=O(\log N)$. The integers $f\left(2^{d}\right)$ and $g\left(2^{d}\right)$ are bounded by $2^{m d}$ and hence the task of multiplying in $\mathcal{R}$ reduces to $O\left(\log ^{2} N\right)$ bit integer multiplication.

Therefore, the complexity of our integer multiplication algorithm $T(N)$ is given by,

$$
\begin{aligned}
T(N) & =O(\mathcal{F}(2 M, k))=O\left(M^{k} \log M \cdot m \cdot \log p+\frac{M^{k} \log M}{\log m} \cdot \mathcal{M}_{\mathcal{R}}\right) \\
& =O\left(N \log N+\frac{N}{\log N \cdot \log \log N} \cdot T\left(O\left(\log ^{2} N\right)\right)\right)
\end{aligned}
$$

Solving the above recurrence leads to the following theorem.
Theorem 4.2. Given two $N$ bit integers, their product can be computed using $N \cdot \log N \cdot 2^{O\left(\log ^{*} N\right)}$ bit operations.

## Choosing the Prime Randomly

To ensure that the search for a prime $p \equiv 1(\bmod 2 M)$ does not affect the overall time complexity of the algorithm, we considered multivariate polynomials to restrict the value of $M$; an alternative is to use randomization.

Proposition 4.3. Assuming ERH, a prime $p \equiv 1(\bmod 2 M)$ can be computed by a randomized algorithm with expected running time $\tilde{O}\left(\log ^{3} M\right)$.

Proof. Titchmarsh Tit30 (see also Tianxin Tia90]) showed, assuming ERH, that the number of primes less than $x$ in the arithmetic progression $\{1+i \cdot 2 M\}_{i>0}$ is given by,

$$
\pi(x, 2 M)=\frac{L i(x)}{\varphi(2 M)}+O(\sqrt{x} \log x)
$$

for $2 M \leq \sqrt{x} \cdot(\log x)^{-2}$, where $L i(x)=\Theta\left(\frac{x}{\log x}\right)$ and $\varphi$ is the Euler totient function. In our case, since $M$ is a power of two, $\varphi(2 M)=M$, and hence for $x \geq 4 M^{2} \cdot \log ^{6} M$, we have $\pi(x, 2 M)=\Omega\left(\frac{x}{M \log x}\right)$. Therefore, for an $i$ chosen uniformly randomly in the range $1 \leq i \leq 2 M \cdot \log ^{6} M$, the probability that $i \cdot 2 M+1$ is a prime is at least $\frac{d}{\log x}$ for a constant $d$. Furthermore, primality test of an $O(\log M)$ bit number can be done in $\tilde{O}\left(\log ^{2} M\right)$ time using Rabin-Miller primality test Mil76. Rab80. Hence, with $x=4 M^{2} \cdot \log ^{6} M$ a suitable prime for our algorithm can be found in expected $O\left(\log ^{3} M\right)$ time.

## 5 A Comparison with Fürer's Algorithm

Our algorithm can be seen as a $p$-adic version of Fürer's integer multiplication algorithm, where the field $\mathbb{C}$ is replaced by $\mathbb{Q}_{p}$, the field of $p$-adic numbers (for a quick introduction, see Baker's online notes Bak07). Much like $\mathbb{C}$, where representing a general element (say in base 2) takes infinitely many bits, representing an element in $\mathbb{Q}_{p}$ takes infinitely many $p$-adic digits. Since we cannot work with infinitely many digits, all arithmetic has to be done with finite precision. Modular arithmetic in the base ring $\mathbb{Z}[\alpha] /\left(p^{c}, \alpha^{m}+1\right)$, can be viewed as arithmetic in the ring $\mathbb{Q}_{p}[\alpha] /\left(\alpha^{m}+1\right)$ keeping a precision of $\varepsilon=p^{-c}$.

Arithmetic with finite precision naturally introduces some errors in computation. However, the nature of $\mathbb{Q}_{p}$ makes the error analysis simpler. The field $\mathbb{Q}_{p}$ comes with a norm $|\cdot|_{p}$ called the $p$-adic norm, which satisfies the stronger triangle inequality $|x+y|_{p} \leq \max \left(|x|_{p},|y|_{p}\right) \quad \operatorname{Bak} 07$, Proposition 2.6]. As a result, unlike in $\mathbb{C}$, the errors in computation do not compound.

Recall that the efficiency of FFT crucially depends on a special principal $2 M$-th root of unity in $\mathbb{Q}_{p}[\alpha] /\left(\alpha^{m}+1\right)$. Such a root is constructed with the help of a primitive $2 M$-th root of unity in $\mathbb{Q}_{p}$. The field $\mathbb{Q}_{p}$ has a primitive $2 M$-th root of unity if and only if $2 M$ divides $p-1$ Bak07, Theorem 5.12]. Also, if $2 M$ divides $p-1$, a $2 M$-th root can be obtained from a $(p-1)$-th root of unity by taking a suitable power. A primitive $(p-1)$-th root of unity in $\mathbb{Q}_{p}$ can be constructed, to sufficient precision, using Hensel Lifting starting from a generator of $\mathbb{F}_{p}^{\times}$.

## 6 Conclusion

As mentioned earlier, there has been two approaches to multiplying integers - one using arithmetic over complex numbers and the other using modular arithmetic. Using complex numbers, Schönhage and Strassen [SS71] gave an $O\left(N \cdot \log N \cdot \log \log N \ldots 2^{O\left(\log ^{*} N\right)}\right)$ algorithm. Fürer |Für07| improved this complexity to $N \cdot \log N \cdot 2^{O\left(\log ^{*} N\right)}$ using some special roots of unity. The other approach, that is modular arithmetic, can be seen as arithmetic in $\mathbb{Q}_{p}$ with certain precision. A direct adaptation of the Schönhage-Strassen's algorithm in the modular setting leads to an $O(N \cdot \log N$. $\log \log N \ldots 2^{O\left(\log ^{*} N\right)}$ ) time algorithm. In this work, we show that by choosing an appropriate prime and a special root of unity, a running time of $N \cdot \log N \cdot 2^{O\left(\log ^{*} N\right)}$ can be achieved through modular arithmetic as well. Therefore, in a way, we have unified the two paradigms. The important question that remains open is:

- Can $N$-bit integers be multiplied using $O(N \cdot \log N)$ bit operations?

Even an improvement of the complexity to $O\left(N \cdot \log N \cdot \log ^{*} N\right)$ operations will be a significant step forward towards answering this question.

## References

[Bak07] Alan J. Baker. An introduction to p-adic numbers and p-adic analysis. Online Notes, 2007. http://www.maths.gla.ac.uk/~ajb/dvi-ps/padicnotes.pdf.
[Für07] Martin Fürer. Faster Integer Multiplication. In STOC, pages 57-66, 2007.
[Für09] Martin Fürer. Faster Integer Multiplication. SIAM J. Comput., 39(3):979-1005, 2009.
[HB92] D. R. Heath-Brown. Zero-free regions for Dirichlet L-functions, and the least prime in an arithmetic progression. In Proceedings of the London Mathematical Society, 64(3), pages 265-338, 1992.
[Isa94] I. Martin Isaacs. Character theory of finite groups. Dover publications Inc., New York, 1994.
[KO63] A Karatsuba and Y Ofman. Multipication of multidigit numbers on automata. English Translation in Soviet Physics Doklady, 7:595-596, 1963.
[Lin44] Yuri V. Linnik. On the least prime in an arithmetic progression, I. The basic theorem, II. The Deuring-Heilbronn's phenomenon. Rec. Math. (Mat. Sbornik), 15:139-178 and 347-368, 1944.
[Mil76] Gary L. Miller. Riemann's Hypothesis and Tests for Primality. J. Comput. Syst. Sci., 13(3):300-317, 1976.
[NZM91] Ivan Niven, Herbert S. Zuckerman, and Hugh L. Montgomery. An Introduction to the Theory of Numbers. John Wiley and Sons, Singapore, 1991.
[Rab80] Michael O. Rabin. Probabilistic algorithm for testing primality. J. Number Theory, 12(1):128-138, 1980.
[Sch82] Arnold Schönhage. Asymptotically Fast Algorithms for the Numerical Multiplication and Division of Polynomials with Complex Coefficients. In Computer Algebra, EUROCAM, volume 144 of Lecture Notes in Computer Science, pages 3-15, 1982.
[Sha99] Igor R. Shafarevich. Basic Notions of Algebra. Springer Verlag, USA, 1999.
[SS71] A Schönhage and V Strassen. Schnelle Multiplikation grosser Zahlen. Computing, 7:281292, 1971.
[Tia90] Cai Tianxin. Primes Representable by Polynomials and the Lower Bound of the Least Primes in Arithmetic Progressions. Acta Mathematica Sinica, New Series, 6:289-296, 1990.
[Tit30] E. C. Titchmarsh. A Divisor Problem. Rend. Circ. Mat. Palerme, 54:414-429, 1930.
[Too63] A L. Toom. The complexity of a scheme of functional elements simulating the multiplication of integers. English Translation in Soviet Mathematics, 3:714-716, 1963.
[Xyl10] Triantafyllos Xylouris. On Linnik's constant. arXiv:abs/0906.2749 [math.NT], 2010.


[^0]:    *A preliminary version appeared in the proceedings of the 40 th ACM Symposium on Theory of Computing, 2008.
    ${ }^{\dagger}$ Research done while the author was at the Dept of Computer Science and Engineering, IIT Kanpur
    ${ }^{\ddagger}$ Research supported through Research I Foundation project NRNM/CS/20030163
    ${ }^{\text {§}}$ Research done while visiting IIT Kanpur under Project FLW/DST/CS/20060225

