The De-Kupur-Saha-Saptharishi algorithm for integer multiplication
The function $\log^{(i)}$ is defined recursively as follows.

- $\log^{(0)} n = n$, and
- $\log^{(i+1)} n = \log_2 \left( \log^{(i)} n \right)$. 

$\log^* n = \min \{ i \geq 0 : \log^{(i)} \leq 1 \}$

The Fürer and the De-Kupur-Saha-Saptharishi algorithms compute the product of two $N$-bit integers in $N \log N 2^{O(\log^* N)}$ bit operations.
Encoding integers by $k$-variate polynomials
We break an $N$-bit integer $a$ into $M^k$ blocks, $\frac{N}{M^k}$ bits in each, which corresponds to representing $a$ in base $q = 2^{\frac{N}{M^k}}$.

Let

$$a = a_{M^k - 1}q^{M^k - 1} + a_{M^k - 2}q^{M^k - 2} + a_0,$$

where $a_i < q$, $i = 0, 1, \ldots, M^k - 1$. 

\[a: \quad a_{M^k - 1} \quad \quad a_1 \quad a_0\]
Next, we convert $a$ into a polynomial.

For this, we express the index $i$ of $a_i$ in base $M$:

$$i = i_k M^{k-1} + i_{k-1} M^{k-2} + \cdots + i_1$$

(note the indices!), where $i_j < M$, $j = 1, 2, \ldots, k$, and encode each term $a_i q^i$ as the monomial

$$a_i X_1^{i_1} X_2^{i_2} \cdots X_k^{i_k}.$$ 

That is, $a$ is converted into the polynomial

$$\sum_{i=0}^{M^k - 1} a_i X_1^{i_1} X_2^{i_2} \cdots X_k^{i_k}.$$ 

The parameters $M$ and $k$ (the latter being a constant) will be chosen so that

$$M^k = O \left( \frac{N}{\log^2 N} \right).$$
Finally, we break each $a_i$ into $\frac{m}{2}$ equal sized blocks, $u = \frac{2N}{mM^k}$ bits in each and encode $a_i$ as a polynomial in $\lambda$ of degree $\frac{m^2}{2}$. That is, if

$$a_i = \sum_{j=0}^{\frac{m^2}{2}-1} a_{i,j} 2^uj,$$

then $a_i$ is encoded by the polynomial

$$\sum_{j=0}^{\frac{m^2}{2}-1} a_{i,j} \lambda^j .$$

Summing up, $a$ is converted into a $k$-variate polynomial $a(X_1, X_2, \ldots, X_k)$ over the ring $\mathbb{Z}(\lambda)/(\lambda^m + 1)$.

The parameter $m$ will be chosen so that

$$m = O(\log N) .$$
Given $N$-bit integers $a$ and $b$, we encode them as polynomials

$$a(X_1, X_2, \ldots, X_k) \quad \text{and} \quad b(X_1, X_2, \ldots, X_k)$$

and compute the product polynomial. The product $ab$ can be recovered by substituting $X_s = q^{M^{s-1}}$, for $s = 1, 2, \ldots, k$, and $\lambda = 2^u$ in the polynomial $a(X_1, X_2, \ldots, X_k)b(X_1, X_2, \ldots, X_k)$.

The coefficients in the product polynomial could be as large as $mM^k 2^{2u}$ (why?).

Thus, it is sufficient to do arithmetic modulo $p^c$, where $p$ is prime and $p^c > mM^k 2^{2u}$.

---

1The choice of the prime $p$ ensures that $c$ is, in fact, a constant.
Lemma  Multiplication of two $N$-bit integers reduces to multiplication of two $k$-variate polynomials, with degree in each variable bounded by $M$, over the ring
\[ R = \mathbb{Z}(\lambda)/(p^c, \lambda^m + 1) \]
for a prime $p$ satisfying $p^c > mM^k 2^u$, where $u = \frac{2N}{mM^k}$.

Furthermore, the reduction can be performed in $O(N)$ time.
Choosing the prime $p$
We shall choose the prime $p$ and $c$ such that the ring $\mathbb{Z}/p^c\mathbb{Z}$ has a principal $2M$th root of unity needed for polynomial multiplication using the discrete Fourier transform.

As we have already seen (where?), in $\mathbb{Z}/p^c\mathbb{Z}$, a $2M$th root of unity is principal if and only if $2M$ divides $p - 1$.

Therefore, we must choose the prime $p$ from the arithmetic progression

$$\{1 + i \cdot 2M : i + 1, 2, \ldots\}.$$  

**Theorem** There exist $\ell$ and $L$ such that for any pair of coprime integers $d$ and $n$, the least prime $p$ such that $p \equiv d \mod n$ is less than $\ell n^{L}$.

\[\text{\textsuperscript{2}}\text{In fact, } L \leq 5.2.\]
Lemma  If \( k > L + 1 \), then \( M^L = O(N^{\frac{L}{L+1}}) \).

Proof Since \( M^k < N \) (why?), we have

\[
M^L < N^{\frac{L}{k}} < N^{\frac{L}{L+1}}.
\]

\( \square \)

Corollary  The least prime \( p \) such that \( p \equiv 1 \mod 2M \) can be found in \( o(N) \) time.

Proof We have to test for primality the first \( O\left(N^{\frac{L}{L+1}}\right) \) integers.

Since each of them is of length \( O(\log N) \) and primality can be tested in polynomial time, the total computation time is

\[
O \left(N^{\frac{L}{L+1}}\right) \text{poly}(\log N) = o(N).
\]

\( \square \)
Finding the root of unity
For what follows, we need a principal $2M$th root of unity $\rho(\lambda)$ in $\mathcal{R}$ such that $(\rho(\lambda))^{M/m} = \lambda$ (why $m$ divides $M$?).

To obtain such a root, we start with computing a $2M$th root of unity $\omega_1$ in $\mathbb{Z}_p$.

For this we compute a generator $\zeta$ of $\mathbb{Z}_p^*$, by successively testing $2, 3, \ldots, p-1$.

Each test can be performed in poly($\log p$) bit operations (why?).

Thus, $\zeta$ can be found in $o(N)$ bit operations (why?)

We put

$$\omega_1 = \zeta^{(p-1)/2M}$$

and compute a principal $2M$th root of unity $\omega_c$ in $\mathbb{Z}_{p^c}$ in additional $o(N)$ bit operations (how?).
Let $\gamma = \omega_c^{M/m}$. Then, $\gamma$ is a principal $2m$th root of unity (why?), implying

$$\lambda^m + 1 = \prod_{i=0}^{m-1} \lambda - \gamma^{2i+1}$$

(why?). Let

$$\rho(\lambda) = \sum_{i=0}^{m-1} \omega_c^{2i+1} \prod_{j=0}^{m-1} \frac{\lambda - \gamma^{2j+1}}{\gamma^{2i+1} - \gamma^{2j+1}}.$$ 

(Why $\gamma^{2i+1} - \gamma^{2j+1} \neq 0 \mod p$?)
\[ \rho(\lambda) = \sum_{i=0}^{m-1} \omega_c^{2i+1} \prod_{\substack{j=0 \atop j \neq i}}^{m-1} \frac{\lambda - \gamma^{2j+1}}{\gamma^{2i+1} - \gamma^{2j+1}} \]

Then, \( \rho(\gamma^{2i+1}) = \omega_c^{2i+1} \) \( i = 0, 1, \ldots, m - 1 \), (why?), implying

\[ (\rho(\gamma^{2i+1}))^M = (\omega_c^{2i+1})^M = (\omega_M^{2i+1}) = (-1)^{2i+1} = -1. \]

Therefore, \( \lambda - \gamma^{2i+1} \) divides \( (\rho(\lambda))^M + 1 \) (why?) and it follows that \( \lambda^m + 1 \) divides \( (\rho(\lambda))^M + 1 \) as well (why?). That is,

\[ (\rho(\lambda))^M + 1 \equiv 0 \pmod{\lambda^m + 1}. \]
\[ \rho \left( \gamma^{2i+1} \right) = \omega_c^{2i+1} \quad i = 0, 1, \ldots, m - 1 \]

In addition,
\[ \left( \rho \left( \gamma^{2i+1} \right) \right)^{M/m} = (\omega_c^{2i+1})^{M/m} = \gamma^{2i+1} \]
(why?).

Therefore,
\[ (\rho(\lambda))^{M/m} = \lambda \mod \lambda - \gamma^{2i+1}, \quad i = 0, 1, \ldots, m - 1 \]
(why?), implying
\[ (\rho(\lambda))^{M/m} = \lambda \mod \lambda^m + 1 \]
(how?).

Note \( \rho(\lambda) \) can be computed in \( \text{poly}(\log N) \) bit operations (why?).
Fourier transform and multiplication of multivariate polynomials
General

Let $S$ be a ring containing a $2M$th principal root of unity $\rho$ and let $P(x) \in S[x]$ be a polynomial of degree less than $2M$. We are going to evaluate $P(x)$ at $1, \rho, \ldots, \rho^{2M-1}$ in two stages as follows.

- **At Stage 1**, for $\gamma = \rho^{M/m}$, we compute the polynomials

  \[
  P_0(x) = \text{rem}(P(x), x^{M/m} - 1) \\
  P_1(x) = \text{rem}(P(x), x^{M/m} - \gamma) \\
  \vdots \\
  P_{2m-1}(x) = \text{rem}(P(x), x^{M/m} - \gamma^{2m-1}).
  \]

  and

- **at Stage 2**, we evaluate each polynomial $P_i(x)$, $i = 0, 1, \ldots, 2m-1$, at the roots of $x^{M/m} - \gamma^i$ (how?).

This will give us with the Fourier transform of the vector of the coefficients of $P(x)$. 
For Stage 1, we have

\[ P(x) = p_0 + \cdots + p_j x^j + \cdots + p_{M-1} x^{M-1} \]

\[ p_\frac{M}{m} x^{\frac{M}{m}} + \cdots + p_{\frac{M}{m}+j} x^{\frac{M}{m}+j} + \cdots + p_{2\frac{M}{m}-1} x^{2\frac{M}{m}-1} \]

\[ \vdots \]

\[ p_\frac{\ell M}{m} x^{\frac{\ell M}{m}} + \cdots + p_{\frac{\ell M}{m}+j} x^{\frac{\ell M}{m}+j} + \cdots + p_{(\ell+1)2\frac{M}{m}-1} x^{(\ell+1)2\frac{M}{m}-1} \]

\[ \vdots \]

\[ p_{2\frac{M}{m}-\frac{M}{m}} x^{2\frac{M}{m}-\frac{M}{m}} + \cdots + p_{2\frac{M}{m}-\frac{M}{m}+j} x^{2\frac{M}{m}-\frac{M}{m}+j} + \cdots + p_{2M-1} x^{2M-1} \]

\[ = \sum_{j=0}^{M-1} x^j \sum_{\ell=0}^{2m-1} p_{\frac{\ell M}{m}+j} x^{\frac{\ell M}{m}} \]
\[
P(x) = \sum_{j=0}^{M/m-1} x^j \sum_{\ell=0}^{2m-1} p_{\ell \frac{M}{m} + j} x^{\ell \frac{M}{m}}
\]

Thus,

\[
P(x) \equiv \sum_{j=0}^{M/m-1} x^j \sum_{\ell=0}^{2m-1} p_{\ell \frac{M}{m} + j} (\gamma^k)^{\ell} \mod x^{M/m} - \gamma^k
\]

(why?).

For a fixed \(j\), the sums \( \sum_{\ell=0}^{2m-1} p_{\ell \frac{M}{m} + j} (\gamma^k)^{\ell} \), \( k = 0, 1, \ldots, 2m - 1 \), are just 2m-point Fourier transforms of \( \left(p_j, p_{\frac{M}{m} + j}, \ldots, p_{(2m-1)\frac{M}{m} + j}\right) \) “at \(\gamma\).”

Thus, the whole computation at Stage 1, can be performed by computing \(M/m\) 2m-point Fourier transforms.

In Stage 2, we have 2m Fourier transforms of dimension \(M/m\) “at \(\rho^m\)” and 2M multiplications in \(S\) by powers of \(\rho\) (which transforms and which multiolications?).
In what follows, $S$ will “recursively” be $S_i = R[X_1, X_2, \ldots, X_{i-1}]$, $i = 1, 2, \ldots, k$, $x$ will be $X_i$, respectively, and $\rho$ will be the principal $2M$th $\rho(\lambda)$ we have computed before (how?). In particular, $\rho^{M/m} = \lambda$.

We shall multiply polynomials $a(X_i)$ and $b(X_i)$, $\deg a(X_i)$, $\deg b(X_i) < M$, over $S_i$. Let $F(M, i)$ denote the bit complexity of this multiplication.

Multiplication of $a(X_i)$ and $b(X_i)$ consists of three $2M$-point Fourier transforms over $S_i$ and $2M$ pointwise multiplications in $S_i$.

Let $D(n, i)$ denote the bit complexity of an $n$-point Fourier transform over $S_i$. It consists of

- the complexity of computing the $2m$ residues at Stage 1;

- the complexity of multiplication of the residue coefficients by the appropriate powers of $\rho$ in Stage 2; and

- computing $2m$ Fourier transforms of dimension $M/m$, also in Stage 2.
Bit complexity of Stage 1
Computing a $2m$-point Fourier transform requires $2m \log_2(2m)$ additions over any $S_i$ and and $m \log_2(2m)$ multiplications of the coefficients of polynomials in $S_i$ by powers of $\lambda$.

Since $\mathcal{R} = \mathbb{Z}[\lambda]/(p^c, \lambda^m + 1)$, multiplication by a power of $\lambda$ of an element in $\mathcal{R}$ is a cyclic shift with subtractions, which takes $O(m \log p)$ bit operations (why?).

An element in $S_i$ is just a polynomial over $\mathcal{R}$ in variables $X_1, X_2, \ldots, X_{i-1}$, with degree in each variable less than $M$. Hence, multiplication by a power of $\lambda$ of an element of $S_i$ can be done in $N_{S_i} = O(M^{i-1}m \log p)$ bit operations (why?).

Therefore, a total of $m \log_2(2m)$ multiplications takes $O((m \log m)N_{S_i})$ bit operations.

It is easy to see that a similar argument (which one?) shows that $2m \log_2(2m)$ additions in $S_i$ also require the same order of time.

Finally, since there are $M/m$ many $2m$-point Fourier transforms, the total time spent in Stage 1 is $O((M \log m)N_{S_i})$ bit operations (why?).
Bit complexity of Stage 2 (what is stage 2?) and the total complexity of the Fourier transform
Suppose that two arbitrary elements in $\mathcal{R}$ can be multiplied using $\mathcal{M}_\mathcal{R}$ bit operations.

Multiplication in $\mathcal{S}_i$ by a power of $\rho$ amounts to $c_{\mathcal{S}_i} = M^{i-1}$ multiplications in $\mathcal{R}$.

Since there are $2M$ such multiplications, the total time is $O(Mc_{\mathcal{S}_i}\mathcal{M}_\mathcal{R})$.

By the defintion of $\mathcal{D}$, the bit complexity of the Fourier transforms at Stage 2 is $2m\mathcal{D}(M/m, i)$.

**Note** \[ (\rho^m)^{(M/m)/m} = \rho^{M/m} = \lambda. \]

Summing up,

\[
\mathcal{D}(M, i) = O(M \log m \cdot N_{\mathcal{S}_i} + M c_{\mathcal{S}_i} M_{\mathcal{R}}) + 2m\mathcal{D}(M/m, i)
\]

\[
= O(M \log m \cdot N_{\mathcal{S}_i} + M c_{\mathcal{S}_i} M_{\mathcal{R}}) \frac{\log M}{\log m}
\]

\[
= O \left( M^i \log M \cdot m \cdot \log p + \frac{M^i \log M}{\log m} M_{\mathcal{R}} \right).
\]
Pointwise multiplications and
total polynomial multiplication time
Finally, the multiplication algorithm performs $2M$ pointwise multiplications in $S$ (what are these multiplications?).

Since elements of $S_i$ are $(i-1)$-variate polynomials over $\mathcal{R}$, with degree in every variable less than $M$, total time taken for pointwise multiplications is $2MF(M, i - 1)$ bit operations.

Therefore,

$$F(M, i) = O\left( M^i \log M \cdot m \log p + \frac{M^i \log M}{\log m} M_{\mathcal{R}} \right) + 2MF(M, i - 1)$$

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

because $i \leq k$ and $k$ is a constant.

Thus,

$$F(M, k) = O\left( M^k \log M \cdot m \log p + \frac{M^k \log M}{\log m} M_{\mathcal{R}} \right).$$
Choosing parameters
\[ F(M, k) = O \left( M^k \log M \cdot m \log p + \frac{M^k \log M}{\log m} M_R \right) \]

- \( k > L + 1 \) (what is \( L \)?)
- \( M^k = O(N/\log^2 N) \) and \( m = O(\log N) \) (therefore, \( M^L = O(N^\varepsilon) \) for some \( \varepsilon < 1 \). Give an upper bound on \( \varepsilon \));
- \( c > 5(k + 1) \); and
- the length of \( p \) is \( O(\log N) \), but \( p^c > 2M^k m 2^{2u} \), where \( u = \frac{2N}{M^k m} \).

Let \( T(n) \) denote the bit complexity of multiplication of \( n \)-bit integers.

**Proposition** \( M_R = T(O(\log^2 N)) \).
Proof Elements of \( R \) can be viewed as polynomials in \( \lambda \) of degree \( m \) (or less) over \( \mathbb{Z}/p^c\mathbb{Z} \). We encode such a polynomial \( f(\lambda) = \sum_{i=0}^{m-1} f_i \lambda^i \), \( f_i < p^c \), \( i = 0, 1, \ldots, m - 1 \), by an integer \( f(2^d) = \sum_{i=0}^{m-1} f_i 2^{di} \), where \( 2^d > mp^{2c} \).

Then the coefficients of the product \( f(\lambda)g(\lambda) \) can be recovered from the integer product \( f(2^d)g(2^d) \) (how? Where have we seen a similar construction?) and the constraint \( 2^d > mp^{2c} \) implies that we can take \( d = O(\log N) \) (why?).

Thus, the length of the encoding is \( O(\log^2 N) \) (why?), implying the the product of the encodings can be computed in \( T(O(\log^2 N)) \) bit operations.

Reducing modulo \( \lambda^m + 1 \) (why do we need to reduce?) can be performed in \( O(md) = O(\log^2 N) \) bit operations (why?), and dividing by \( p^c \), which has \( O(\log N) \) bits, can be performed in \( T(O(\log N)) \) bit operations (why? Why do we have to deivide by \( p^c \)).
Summing up,

\[ M_R = T \left( O \left( \log^2 N \right) \right) + O(\log N \cdot T(O(\log N))) = T \left( O \left( \log^2 N \right) \right) \]

(why?)

\[ T(N) = F(M, k) = O \left( M^k \log M \cdot m \log p + \frac{M^k \log M}{\log m} M_R \right) \]

\[ = O \left( N \log N + \frac{N}{\log N \log \log N} T \left( O \left( \log^2 N \right) \right) \right). \]

After solving the recurrence from the corollary at home, we obtain the desired upper bound.

**Theorem**  \( T(N) = N \log N \ 2^{O(\log^* N)} \) (where is the \( O \)?)