How to multiply big integers

Standard idea: Use polynomial with coefficients in \{0, 1, \ldots, 9\} to represent integer in radix 10.

Example of representation: 
\[ 839 = 8 \cdot 10^2 + 3 \cdot 10^1 + 9 \cdot 10^0 = \text{value (at } t = 10) \text{ of polynomial } 8t^2 + 3t^1 + 9t^0. \]

Convenient to express polynomial inside computer as array \[ 9, 3, 8 \] (or \[ 9, 3, 8, 0 \] or \[ 9, 3, 8, 0, 0 \] or \ldots ): “\( p[0] = 9; \ p[1] = 3; \ p[2] = 8 \)”

Multiply two integers by multiplying polynomials that represent the integers.

Polynomial multiplication involves \textit{small} integer coefficients. Have split one big multiplication into many small operations.

Example, squaring 839:
\[
(8t^2 + 3t^1 + 9t^0)^2 =
8t^2(8t^2 + 3t^1 + 9t^0) +
3t^1(8t^2 + 3t^1 + 9t^0) +
9t^0(8t^2 + 3t^1 + 9t^0) =
64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0.
\]
How to multiply big integers

Standard idea: Use polynomial with coefficients in \( \{0, 1, \ldots, 9\} \) to represent integer in radix 10.

Example of representation:

\[839 = 8 \cdot 10^2 + 3 \cdot 10^1 + 9 \cdot 10^0 = 8t^2 + 3t^1 + 9t^0.\]

Convenient to express polynomial inside computer as array 9, 3, 8 (or 9, 3, 8, 0 or 9, 3, 8, 0, 0 or \ldots):

\[p[0] = 9; \ p[1] = 3; \ p[2] = 8.\]

Multiply two integers

by multiplying polynomials that represent the integers.

Polynomial multiplication involves \textit{small} integer coefficients.

Have split one big multiplication into many small operations.

Example, squaring 839:

\[(8t^2 + 3t^1 + 9t^0)^2 =
8t^2(8t^2 + 3t^1 + 9t^0) +
3t^1(8t^2 + 3t^1 + 9t^0) +
9t^0(8t^2 + 3t^1 + 9t^0) =
64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0.\]

Oops, product polynomial usually has coefficients > 9.
So “carry” extra digits:

\[ct^j \rightarrow \lfloor c = 10 \rfloor t^{j+1} + (c \mod 10) t^j.\]

Example, squaring 839:

\[64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0;
64t^4 + 48t^3 + 62t^1 + 1t^0;
64t^4 + 159t^2 + 2t^1 + 1t^0;
64t^4 + 63t^3 + 9t^2 + 2t^1 + 1t^0;
70t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0;
7t^5 + 0t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0.\]

In other words, 839

\[\times 839 = 703921.\]
Multiply two integers by multiplying polynomials that represent the integers. Polynomial multiplication involves small integer coefficients. Have split one big multiplication into many small operations.

Example, squaring 839:
\[(8t^2 + 3t^1 + 9t^0)^2 =
8t^2(8t^2 + 3t^1 + 9t^0) +
3t^1(8t^2 + 3t^1 + 9t^0) +
9t^0(8t^2 + 3t^1 + 9t^0) =
64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0.
\]

Oops, product polynomial usually has coefficients > 9. So “carry” extra digits:
\[ct^j \rightarrow \lfloor c/10 \rfloor t^{j+1} + (c \mod 10) t^j.\]

Example, squaring 839:
\[64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0\]
\[70t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0.\]

In other words, 839
\[839^2 = 703921.\]
Multiply two integers by multiplying polynomials that represent the integers.

Polynomial multiplication involves small integer coefficients.

Example, squaring 839:

\[
(8t^2 + 3t^1 + 9t^0)^2 = \\
8t^2(8t^2 + 3t^1 + 9t^0) + \\
3t^1(8t^2 + 3t^1 + 9t^0) + \\
9t^0(8t^2 + 3t^1 + 9t^0) = \\
64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0.
\]

Oops, product polynomial usually has coefficients > 9. So “carry” extra digits:

\[
ct^j \rightarrow \lfloor c/10 \rfloor t^{j+1} + (c \mod 10) t^j.
\]

Example, squaring 839:

\[
64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0 = \\
64t^4 + 48t^3 + 153t^2 + 62t^1 + 81t^0 = \\
64t^4 + 48t^3 + 159t^2 + 2t^1 + 81t^0 = \\
64t^4 + 63t^3 + 9t^2 + 2t^1 + 81t^0 = \\
70t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0 = \\
7t^5 + 0t^4 + 3t^3 + 9t^2 + 2t^1.
\]

In other words, \(839^2 = 703921\).
Multiply two integers by multiplying polynomials that represent the integers. Polynomial multiplication involves small integer coefficients. Have split one big multiplication into many small operations.

Example, squaring 839:

\[(8t^2 + 3t^1 + 9t^0)^2 =
8t^2(8t^2 + 3t^1 + 9t^0) + 
3t^1(8t^2 + 3t^1 + 9t^0) + 
9t^0(8t^2 + 3t^1 + 9t^0) = 
64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0.\]

Oops, product polynomial usually has coefficients > 9. So “carry” extra digits:

\[ct^j \rightarrow \lfloor c/10 \rfloor t^{j+1} + (c \mod 10) t^j.\]

Example, squaring 839:

\[64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0;\]
\[64t^4 + 48t^3 + 153t^2 + 62t^1 + 1t^0;\]
\[64t^4 + 48t^3 + 159t^2 + 2t^1 + 1t^0;\]
\[64t^4 + 63t^3 + 9t^2 + 2t^1 + 1t^0;\]
\[70t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0;\]
\[7t^5 + 0t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0.\]

In other words, \(839^2 = 703921\).
Multiply two integers by multiplying polynomials that represent the integers.

Polynomial multiplication involves small integer coefficients.

Have split one big multiplication into many small operations.

Example, squaring 839:

\[(8t^2 + 3t + 9)^2 = (8t^2 + 3t^1 + 9t^0)^2 = 64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0.\]

Oops, product polynomial usually has coefficients > 9. So “carry” extra digits:

\[ct^j \rightarrow \lfloor c/10 \rfloor t^{j+1} + (c \mod 10)t^j.\]

Example, squaring 839:

\[64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0;\]
\[64t^4 + 48t^3 + 153t^2 + 62t^1 + 1t^0;\]
\[64t^4 + 48t^3 + 159t^2 + 2t^1 + 1t^0;\]
\[64t^4 + 63t^3 + 9t^2 + 2t^1 + 1t^0;\]
\[70t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0;\]
\[7t^5 + 0t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0.\]

In other words, 839^2 = 703921.
Oops, product polynomial usually has coefficients > 9. So “carry” extra digits:

\[ c t^j \rightarrow \lfloor c/10 \rfloor t^{j+1} + (c \mod 10) t^j. \]

Example, squaring 839:

\[ 64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0; \]
\[ 64t^4 + 48t^3 + 153t^2 + 62t^1 + 1t^0; \]
\[ 64t^4 + 48t^3 + 159t^2 + 2t^1 + 1t^0; \]
\[ 64t^4 + 63t^3 + 9t^2 + 2t^1 + 1t^0; \]
\[ 70t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0; \]
\[ 7t^5 + 0t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0. \]

In other words, 839^2 = 703921.
Oops, product polynomial usually has coefficients $> 9$. So “carry” extra digits:

$$ct^j \rightarrow \lfloor c/10 \rfloor t^{j+1} + (c \mod 10)t^j.$$ 

Example, squaring 839:

$$64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0;$$
$$64t^4 + 48t^3 + 153t^2 + 62t^1 + 1t^0;$$
$$64t^4 + 48t^3 + 159t^2 + 2t^1 + 1t^0;$$
$$64t^4 + 63t^3 + 9t^2 + 2t^1 + 1t^0;$$
$$70t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0;$$
$$7t^5 + 0t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0.$$ 

In other words, $839^2 = 703921$. 

What operations were used?
Oops, product polynomial usually has coefficients $> 9$. So “carry” extra digits:
\[ ct^j \rightarrow \lfloor c/10 \rfloor t^{j+1} + (c \mod 10) t^j. \]

Example, squaring $839$:
\[
\begin{align*}
64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0; \\
64t^4 + 48t^3 + 153t^2 + 62t^1 + 1t^0; \\
64t^4 + 48t^3 + 159t^2 + 2t^1 + 1t^0; \\
64t^4 + 63t^3 + 9t^2 + 2t^1 + 1t^0; \\
70t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0; \\
7t^5 + 0t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0.
\end{align*}
\]

In other words, $839^2 = 703921$. 

What operations were used here?
Oops, product polynomial usually has coefficients > 9. So “carry” extra digits: 
\[ c_t^j \rightarrow \lfloor \frac{c_t^j}{10} \rfloor t_j^{j+1} + (c_t^j \mod 10) t_j^j. \]

Example, squaring 839:

\[ 8^3 + 153t^2 + 54t^1 + 81t^0; \]
\[ 48^3 + 153t^2 + 62t^1 + 1t^0; \]
\[ 48^3 + 159t^2 + 2t^1 + 1t^0; \]
\[ 63^3 + 9t^2 + 2t^1 + 1t^0; \]
\[ 83^3 + 9t^2 + 2t^1 + 1t^0; \]
\[ 3^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0. \]

In other words, 839^2 = 703921.
Polyomino coefficients $> 9$.

Digits:
\[ t^2 + 54t^1 + 81t^0; \]
\[ t^2 + 62t^1 + 1t^0; \]
\[ 9t^2 + 2t^1 + 1t^0; \]
\[ 9t^2 + 2t^1 + 1t^0; \]
\[ 9t^2 + 2t^1 + 1t^0. \]

So $9^2 = 703921$.
Oops, product polynomial usually has coefficients > 9. So "carry" extra digits: $ct_j \rightarrow \lfloor c = 10 \rfloor t_{j+1} + (c \mod 10) t_j$.

Example, squaring 839:

$64 t_4 + 48 t_3 + 153 t_2 + 54 t_1 + 81 t_0$;

$64 t_4 + 48 t_3 + 159 t_2 + 2 t_1 + 1 t_0$;

$64 t_4 + 63 t_3 + 9 t_2 + 2 t_1 + 1 t_0$;

$70 t_4 + 3 t_3 + 9 t_2 + 2 t_1 + 1 t_0$;

$7 t_5 + 0 t_4 + 3 t_3 + 9 t_2 + 2 t_1 + 1 t_0$.

In other words, $839^2 = 703921$.

What operations were used here?

- Multiply
- Add
- Divide by 10
- Mod 10

703921.
What operations were used here?

8 \rightarrow 3 \rightarrow 9 \rightarrow \text{multiply} \rightarrow 72 \rightarrow 9 \rightarrow 72 \rightarrow \text{add} \rightarrow 153 \rightarrow \text{add} \rightarrow 159 \rightarrow \text{add} \rightarrow \text{divide by 10} \rightarrow 15 \rightarrow \text{mod 10} \rightarrow 9
The scaled variation

\[ 839 = 800 + 30 + 9 = \text{value (at } t = 1) \text{ of polynomial} \]

\[ 800 t^2 + 30 t + 9. \]

**Squaring:**

\[ (800 t^2 + 30 t + 9)^2 = 640000 t^4 + 48000 t^3 + 15300 t^2 + 540 t + 81. \]

**Carrying:**

\[ 640000 t^4 + 48000 t^3 + 15300 t^2 + 540 t \leftarrow 620 t^1 + 1 t^0; \ldots \]

\[ 700000 t^5 + 0 t^4 + 3000 t^3 + 900 t^2 + 20 t^1 + 1 t^0. \]
The scaled variation

\[ 839 = 800 + 30 + 9 = \text{value (at } t = 1) \text{ of polynomial } 800t^2 + 30t^1 + 9t^0. \]

Squaring: \((800t^2 + 30t^1 + 9t^0)^2 = 640000t^4 + 48000t^3 + 15300t^2 + 540t^1 + 81t^0.\)

Carrying:
\[ 640000t^4 + 48000t^3 + 15300t^2 + 540t^1 + 81t^0; \]
\[ 640000t^4 + 48000t^3 + 15300t^2 + 540t^1 + \text{t}^0; \]
\[ 700000t^5 + 0t^4 + 3000t^3 + 900t^2 + 20t^1 + \text{t}^0. \]
The scaled variation

\[ 839 = 800 + 30 + 9 = \]

value (at \( t = 1 \)) of polynomial

\[ 800t^2 + 30t^1 + 9t^0. \]

Squaring: \((800t^2 + 30t^1 + 9t^0)^2 = \]

\[ 640000t^4 + 48000t^3 + 15300t^2 + 540t^1 + 81t^0. \]

Carrying:

\[ 640000t^4 + 48000t^3 + 15300t^2 + 620t^1 + 1t^0; \]

\[ 700000t^5 + 0t^4 + 3000t^3 + 9t^2 + 20t^1 + 1t^0. \]
The scaled variation

\[ 839 = 800 + 30 + 9 = \]
value (at \( t = 1 \)) of polynomial
\[ 800t^2 + 30t^1 + 9t^0. \]

Squaring: \((800t^2+30t^1+9t^0)^2 = 640000t^4 + 48000t^3 + 15300t^2 + 540t^1 + 81t^0.\]

Carrying:
\[ 640000t^4 + 48000t^3 + 15300t^2 + 540t^1 + 81t^0; \]
\[ 640000t^4 + 48000t^3 + 15300t^2 + 620t^1 + 1t^0; \]
\[ 700000t^5 + 0t^4 + 3000t^3 + 900t^2 + 20t^1 + 1t^0. \]
The scaled variation

\[ 839 = 800 + 30 + 9 = \]

value (at \( t = 1 \)) of polynomial \( 800t^2 + 30t^1 + 9t^0 \).

Squaring: \((800t^2 + 30t^1 + 9t^0)^2 = 640000t^4 + 48000t^3 + 15300t^2 + 540t^1 + 81t^0 \).

Carrying:

\[ 640000t^4 + 48000t^3 + 15300t^2 + 540t^1 + 81t^0; \]
\[ 640000t^4 + 48000t^3 + 15300t^2 + 620t^1 + 1t^0; \]
\[ 700000t^5 + 0t^4 + 3000t^3 + 900t^2 + 20t^1 + 1t^0. \]
The scaled variation
\[ 839 = 800 + 30 + 9 = \]
value (at \( t = 1 \)) of polynomial
\[ 800t^2 + 30t^1 + 9t^0. \]

Squaring:
\[ (800t^2 + 30t^1 + 9t^0)^2 = 640000t^4 + 48000t^3 + 15300t^2 + 540t^1 + 81t^0. \]

Carrying:
\[ 640000t^4 + 48000t^3 + 15300t^2 + 620t^1 + 1t^0; \]
\[ 640000t^4 + 48000t^3 + 15300t^2 + 620t^1 + 1t^0; \]
\[ 700000t^5 + 0t^4 + 3000t^3 + 900t^2 + 20t^1 + 1t^0. \]
The scaled variation

$$839 = 800 + 30 + 9 =$$

value (at $$t = 1$$) of polynomial

$$800t^2 + 30t^1 + 9t^0.$$  

Squaring:  

$$(800t^2 + 30t^1 + 9t^0)^2 =$$

$$640000t^4 + 48000t^3 + 15300t^2 + 540t^1 + 81t^0.$$  

Carrying:

$$640000t^4 + 48000t^3 + 15300t^2 + 540t^1 + 81t^0;$$

$$640000t^4 + 48000t^3 + 15300t^2 + 620t^1 + 1t^0; \ldots$$

$$700000t^5 + 0t^4 + 3000t^3 + 900t^2 + 20t^1 + 1t^0.$$  

What operations were used here?
The scaled variation

\[ 839 = 800 + 30 + 9 = \]
value (at \( t = 1 \)) of polynomial \( 800t^2 + 30t^1 + 9t^0 \).

Squaring: \((800t^2 + 30t^1 + 9t^0)^2 = 640000t^4 + 48000t^3 + 15300t^2 + 540t^1 + 81t^0\).

Carrying:
\[
640000t^4 + 48000t^3 + 15300t^2 + 540t^1 + 81t^0;
\]
\[
640000t^4 + 48000t^3 + 15300t^2 + 620t^1 + 1t^0; \quad \ldots
\]
\[
700000t^5 + 0t^4 + 3000t^3 + 900t^2 + 20t^1 + 1t^0.
\]

What operations were used here?
The scaled variation

800 + 30 + 9 = value (at t = 1) of polynomial

\[ 800t^2 + 30t^1 + 9t^0. \]

\[ g: \ (800t^2+30t^1+9t^0)^2 = \]

\[ 800 \]

\[ \downarrow \]

\[ 7200 \]

\[ \downarrow \]

\[ 900 \]

\[ \downarrow \]

\[ 7200 \]

\[ \text{multiply} \]

\[ 9 \]

\[ \downarrow \]

\[ 15300 \]

\[ \text{add} \]

\[ 30 \]

\[ \downarrow \]

\[ 600 \]

\[ \text{add} \]

\[ 15900 \]

\[ \text{subtract} \]

\[ 15000 \]

\[ \text{mod 1000} \]

\[ 900 \]

\[ 900 \]

\[ 600 \]

\[ \cdots \]

\[ 15300 \]

\[ 7200 \]

\[ 7200 \]

\[ 15300 \]

\[ 15900 \]

\[ 15000 \]

\[ 5 \text{ mults, 4 adds.} \]

\[ (\cdots + f_2 t^2 + f_1 t^1 + f_0 t^0)^2 \]

\[ \text{has coefficients such as} \]

\[ f_4 f_0 + f_3 f_1 + f_2 f_2 + f_1 f_3 + f_0 f_4. \]

\[ 5 \text{ mults, 4 adds.} \]
The scaled variation 
839 = 800 + 30 + 9 =
value (at \( t = 1 \)) of polynomial 
\( 800t^2 + 30t + 9 \).

Squaring: 
\[(800t^2 + 30t + 9)^2 = 640000t^4 + 48000t^3 + 15300t^2 + 540t + 81 \].

Carrying:
\[
\begin{align*}
640000t^4 &+ 48000t^3 + 15300t^2 + 540t + 81 \\
&\quad + 7200t^4 + 900t^2 + 600t + 900 \\
&\quad + 15900t^2 + 15900t + 600 \\
&\quad = 800000t^4 + 810000t^3 + 328200t^2 + 31500t + 9099. \\
&\quad \mod 1000.
\end{align*}
\]

Speedup: double inside squaring 
\((\cdots + f_2 t^2 + f_1 t + f_0)^2 \) has coefficients such as 
\( f_4 f_0 + f_3 f_1 + f_2 f_2 + f_1 f_3 + f_0 f_4 \).

5 mults, 4 adds.
The scaled variation \( 839 = 800 + 30 + 9 \) is the value (at \( t = 1 \)) of polynomial \( 800 t^2 + 30 t + 9 \).

Squaring: \( (800 t^2 + 30 t + 9)^2 = 640000 t^4 + 48000 t^3 + 15300 t^2 + 620 t^1 + 1 t^0 \); \( 640000 t^4 + 48000 t^3 + 15300 t^2 + 620 t^1 + 1 t^0 ; \ldots \); \( 700000 t^5 + 0 t^4 + 3000 t^3 + 900 t^2 + 20 t^1 + 1 t^0 \).

What operations were used here?

\[
\begin{align*}
800 &\quad \downarrow \quad \downarrow \quad \rightarrow \quad \rightarrow \\
7200 &\quad \rightarrow \quad \rightarrow \\
900 &\quad \downarrow \quad \downarrow \\
7200 &\quad \downarrow \\
15300 &\quad \downarrow \quad \text{multiply} \\
15900 &\quad \downarrow \quad \text{add} \\
15000 &\quad \downarrow \quad \text{subtract} \\
900 &\quad \downarrow \quad \text{mod 1000} \\
600 &\quad \downarrow \quad \text{add} \\
\end{align*}
\]

Speedup: double inside squaring \((\cdots + f_2 t^2 + f_1 t^1 + f_0 t^0)^2\) has coefficients such as \( f_4 f_0 + f_3 f_1 + f_2 f_2 + f_1 f_3 + f_0 f_4 \). 5 mults, 4 adds.
What operations were used here?

800
↓
7200

30
↓
900

9
multiply
↓
7200

15300
add
↓
600

15900
add
↓
mod 1000

15000
subtract

Speedup: double inside squaring

\((\cdots + f_2 t^2 + f_1 t^1 + f_0 t^0)^2\)

has coefficients such as

\(f_4 f_0 + f_3 f_1 + f_2 f_2 + f_1 f_3 + f_0 f_4\).

5 mults, 4 adds.
What operations were used here?

800
→
7200
↓
900
→
7200
↓
multiply

9
→

 multiply

15300
↓
add

900
→
7200
↓

 add

15900
↓
add

600
→

 add

15900
↓
subtract

900
→
mod 1000

15000
→

mod 1000

7200

Speedup: double inside squaring

\((\cdots + f_2 t^2 + f_1 t^1 + f_0 t^0)^2\)

has coefficients such as

\[ f_4 f_0 + f_3 f_1 + f_2 f_2 + f_1 f_3 + f_0 f_4. \]

5 mults, 4 adds.

Compute more efficiently as

\[ 2f_4 f_0 + 2f_3 f_1 + f_2 f_2. \]

3 mults, 2 adds, 2 doublings.

Save \(\approx 1/2\) of the mults if there are many coefficients.
What operations were used here?

Speedup: double inside squaring

\((\cdots + f_2 t^2 + f_1 t^1 + f_0 t^0)^2\)

has coefficients such as

\(f_4 f_0 + f_3 f_1 + f_2 f_2 + f_1 f_3 + f_0 f_4\).

5 mults, 4 adds.

Compute more efficiently as

\(2(f_4 f_0 + f_3 f_1) + f_2 f_2\).

3 mults, 2 adds, 2 doublings.

Save \(\approx 1/2\) of the mults if there are many coefficients.

Faster alternative:

\(2(f_4 f_0 + f_3 f_1) + f_2 f_2\).

3 mults, 2 adds, 1 doubling.

Save \(\approx 1/2\) of the adds if there are many coefficients.
What operations were used here?

Speedup: double inside squaring

\[(\cdots + f_2 t^2 + f_1 t^1 + f_0 t^0)^2\]

has coefficients such as

\[f_4 f_0 + f_3 f_1 + f_2 f_2 + f_1 f_3 + f_0 f_4.\]

5 mults, 4 adds.

Compute more efficiently as

\[2f_4 f_0 + 2f_3 f_1 + f_2 f_2.\]

3 mults, 2 adds, 2 doublings.

Save \(\approx 1/2\) of the mults
if there are many coefficients.

Faster alternative:

\[2(f_4 f_0 + f_3 f_1) + f_2 f_2.\]

3 mults, 2 adds, 1 doubling.

Save \(\approx 1/2\) of the adds
if there are many coefficients.
What operations were used here?

Speedup: double inside squaring

\[(\cdots + f_2 t^2 + f_1 t^1 + f_0 t^0)^2\]

has coefficients such as

\[f_4 f_0 + f_3 f_1 + f_2 f_2 + f_1 f_3 + f_0 f_4.\]

5 mults, 4 adds.

Compute more efficiently as

\[2f_4 f_0 + 2f_3 f_1 + f_2 f_2.\]

3 mults, 2 adds, 2 doublings.

Save \(\approx 1/2\) of the mults if there are many coefficients.

Faster alternative:

\[2(f_4 f_0 + f_3 f_1) + f_2 f_2.\]

3 mults, 2 adds, 1 doubling.

Save \(\approx 1/2\) of the adds if there are many coefficients.
Speedup: double inside squaring

\[(\cdots + f_2 t^2 + f_1 t^1 + f_0 t^0)^2\]

has coefficients such as

\[f_4 f_0 + f_3 f_1 + f_2 f_2 + f_1 f_3 + f_0 f_4.\]

5 mults, 4 adds.

Compute more efficiently as

\[2f_4 f_0 + 2f_3 f_1 + f_2 f_2.\]

3 mults, 2 adds, 2 doublings.

Save \(\approx 1/2\) of the mults if there are many coefficients.

Faster alternative:

\[2(f_4 f_0 + f_3 f_1) + f_2 f_2.\]

3 mults, 2 adds, 1 doubling.

Save \(\approx 1/2\) of the adds if there are many coefficients.
Speedup: double inside squaring

\[(\cdots + f_2 t^2 + f_1 t^1 + f_0 t^0)^2\]

has coefficients such as
\[f_4 f_0 + f_3 f_1 + f_2 f_2 + f_1 f_3 + f_0 f_4.\]

5 mults, 4 adds.

Compute more efficiently as
\[2f_4 f_0 + 2f_3 f_1 + f_2 f_2.\]

3 mults, 2 adds, 2 doublings.

Save \(\approx 1/2\) of the mults
if there are many coefficients.

Faster alternative:
\[2(f_4 f_0 + f_3 f_1) + f_2 f_2.\]

3 mults, 2 adds, 1 doubling.

Save \(\approx 1/2\) of the adds
if there are many coefficients.

Even faster alternative:
\[(2f_0)f_4 + (2f_1)f_3 + f_2 f_2,\]

after precomputing \(2f_0, 2f_1, \ldots\).

3 mults, 2 adds, 0 doublings.

Precomputation \(\approx 0.5\) doublings.
Faster alternative:
\[ 2(f_4 f_0 + f_3 f_1) + f_2 f_2. \]
3 mults, 2 adds, 1 doubling.

Even faster alternative:
\[ (2f_0)f_4 + (2f_1)f_3 + f_2 f_2, \]
after precomputing \( 2f_0, 2f_1, \ldots \).

3 mults, 2 adds, 0 doublings.
Precomputation \( \approx 0.5 \) doublings.
Faster alternative:
\[2(f_4 f_0 + f_3 f_1) + f_2 f_2,\]
3 mults, 2 adds, 1 doubling.

Save \(\approx 1/2\) of the adds
if there are many coefficients.

Even faster alternative:
\[(2f_0) f_4 + (2f_1) f_3 + f_2 f_2,\]
after precomputing \(2f_0, 2f_1, \ldots\)
3 mults, 2 adds, 0 doublings.
Precomputation \(\approx 0.5\) doublings.

Speedup: allow negative coeffs
Recall \(159 \mapsto 15, 9\).
Scaled: \(15900 \mapsto 15000, 900\).
Alternative: \(159 \mapsto 16, -1\).
Scaled: \(15900 \mapsto 16000, -100\).
Use digits \(\{-5, -4, \ldots, 4, 5\}\)
instead of \(\{0, 1, \ldots\}\).
Small disadvantage: need \(-\).
Several small advantages:
easily handle negative integers;
easily handle subtraction;
reduce products a bit.
Faster alternative:
$$2(f_4f_0 + f_3f_1) + f_2f_2.$$  
3 mults, 2 adds, 1 doubling.

Save $\approx 1/2$ of the adds  
if there are many coefficients.

Even faster alternative:
$$ (2f_0)f_4 + (2f_1)f_3 + f_2f_2,$$
after precomputing $2f_0, 2f_1, \ldots$.

3 mults, 2 adds, 0 doublings.  
Precomputation $\approx 0.5$ doublings.

Speedup: allow negative coeffs
Recall $159 \mapsto 15, 9$.
Scaled: $15900 \mapsto 15000, 900$.

Alternative: $159 \mapsto 16, -1$.
Scaled: $15900 \mapsto 16000, -100$.

Use digits $\{ -5, -4, \ldots, 4, 5 \}$  
instead of $\{0, 1, \ldots, 9\}$.
Small disadvantage: need $-\cdot$.
Several small advantages:  
easily handle negative integers;  
easily handle subtraction;  
reduce products a bit.
Faster alternative:
\[2(f_4f_0 + f_3f_1) + f_2f_2.\]
3 mults, 2 adds, 1 doubling.

Save \(\approx 1/2\) of the adds if there are many coefficients.

Even faster alternative:
\[(2f_0)f_4 + (2f_1)f_3 + f_2f_2,\]
after precomputing \(2f_0, 2f_1, \ldots\).
3 mults, 2 adds, 0 doublings.
Precomputation \(\approx 0.5\) doublings.

Speedup: allow negative coeffs
Recall 159 ↦→ 15, 9.
Scaled: 15900 ↦→ 15000, 900.

Alternative: 159 ↦→ 16, −1.
Scaled: 15900 ↦→ 16000, −100.

Use digits \((-5, -4, \ldots, 4, 5)\)
instead of \(\{0, 1, \ldots, 9\}\).
Small disadvantage: need \(-\).
Several small advantages:
easily handle negative integers;
easily handle subtraction;
reduce products a bit.
Faster alternative: 
\[ (f_3 f_1) + f_2 f_2. \]
2 adds, 1 doubling.

1/2 of the adds 
are many coefficients.

Better alternative: 
\[- (2f_1)f_3 + f_2 f_2, \]
recomputing \(2f_0, 2f_1, \ldots\).

2 adds, 0 doublings.
Computation \(\approx 0.5\) doublings.

---

Speedup: allow negative coeffs
Recall \(159 \mapsto 15, 9.\)
Scaled: \(15900 \mapsto 15000, 900.\)

Alternative: \(159 \mapsto 16, -1.\)
Scaled: \(15900 \mapsto 16000, -100.\)

Use digits \((-5, -4, \ldots, 4, 5)\) 
instead of \(\{0, 1, \ldots, 9\}\).

Small disadvantage: need \(-\).

Several small advantages:
easily handle negative integers;
easily handle subtraction;
reduce products a bit.

---

Speedup: delay carries
Computing (e.g.) big \(ab + c^2\): 
multiply \(a; b\) polynomials, carry,
square \(c\) poly, carry, add, carry.

\[ (3t^2 + 1t^1 + 4t^0)(2t^2 + 7t^1 + 1t^0) = 6t^4 + 23t^3 + 18t^2 + 29t^1 + 4t^0; \]

\[ \text{carry: } 8t^4 + 5t^3 + 0t^2 + 9t^1 + 4t^0. \]

As before \(7t^5 + \ldots\):
\[ 64t^4 + 4t^3 + 0t^2 + 1t^1 + 5t^0; \]
\[ +: 7t^5 + \ldots; \]
\[ 7t^5 + 8t^4 + \ldots. \]
Speedup: allow negative coeffs

Recall 159 $\mapsto$ 15, 9.
Scaled: 15900 $\mapsto$ 15000, 900.

Alternative: 159 $\mapsto$ 16, $-1$.
Scaled: 15900 $\mapsto$ 16000, $-100$.

Use digits $\{-5, -4, \ldots, 4, 5\}$ instead of $\{0, 1, \ldots, 9\}$.
Small disadvantage: need $-$. Several small advantages:
easily handle negative integers;
easily handle subtraction;
reduce products a bit.

Speedup: delay carries
Computing (e.g.) big $ab + c^2$:
multiply $a, b$ polynomials, carry;
square $c$ poly, carry, add, carry.

\[ (3t^2 + t^1 + 4t^0)(2t^2 + 7t^1 + t^0) = 6t^4 + 23t^3 + 18t^2 \]
carry: $8t^4 + 5t^3 + \ldots$
As before $\ldots$
Speedup: allow negative coeffs

Recall 159 ↦→ 15, 9.
Scaled: 15900 ↦→ 15000, 900.
Alternative: 159 ↦→ 16, −1.
Scaled: 15900 ↦→ 16000, −100.

Use digits \{-5, -4, \ldots, 4, 5\} instead of \{0, 1, \ldots, 9\}.
Small disadvantage: need −.
Several small advantages:
easily handle negative integers;
easily handle subtraction;
reduce products a bit.

Speedup: delay carries

Computing (e.g.) big \(ab + c^2\):
multiply \(a, b\) polynomials, carry,
square \(c\) poly, carry, add, carry.
e.g. \(a = 314, b = 271, c = 839\):
\[(3t^2 + 1t^1 + 4t^0)(2t^2 + 7t^1 + 1t^0) =
6t^4 + 23t^3 + 18t^2 + 29t^1 + 8t^0;\]
carry: \(8t^4 + 5t^3 + 0t^2 + 9t^1\).

As before \((8t^2 + 3t^1 + 9t^0)^2 =
64t^4 + 48t^3 + 153t^2 + 54t^1 + 7t^0;\)
\(+: 7t^5 + 8t^4 + 8t^3 + 9t^2 + 11t^1;\)
\(7t^5 + 8t^4 + 9t^3 + 0t^2 + 1t^1\)}
Speedup: allow negative coeffs

Recall 159 ↦ 15, 9.
Scaled: 15900 ↦→ 15000, 900.

Alternative: 159 ↦→ 16, −1.
Scaled: 15900 ↦→ 16000, −100.

Use digits \{-5, −4, \ldots, 4, 5\} instead of \{0, 1, \ldots, 9\}.
Small disadvantage: need −.
Several small advantages:
easily handle negative integers;
easily handle subtraction;
reduce products a bit.

Speedup: delay carries

Computing (e.g.) big \(ab + c^2\):
multiply \(a, b\) polynomials, carry,
square \(c\) poly, carry, add, carry.

E.g. \(a = 314, b = 271, c = 839\):
\[(3t^2+1t^1+4t^0)(2t^2+7t^1+1t^0) = 6t^4 + 23t^3 + 18t^2 + 29t^1 + 4t^0;\]
carry: \(8t^4 + 5t^3 + 0t^2 + 9t^1 + 4t^0\).

As before \((8t^2 + 3t^1 + 9t^0)^2 = 64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0;\)
\(7t^5 + 0t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0.\)

+: \(7t^5 + 8t^4 + 8t^3 + 9t^2 + 11t^1 + 5t^0;\)
\(7t^5 + 8t^4 + 9t^3 + 0t^2 + 1t^1 + 5t^0.\)
Speedup: allow negative coeffs

- Speedup: delay carries

Computing (e.g.) big $ab + c^2$:
- Multiply $a, b$ polynomials, carry,
- Square $c$ poly, carry, add, carry.

**Example:** $a = 314, b = 271, c = 839$:

$$(3t^2 + t^1 + 4t^0)(2t^2 + 7t^1 + 1t^0) = 6t^4 + 23t^3 + 18t^2 + 29t^1 + 4t^0;$$

- Carry: $8t^4 + 5t^3 + 0t^2 + 9t^1 + 4t^0$.  

As before $(8t^2 + 3t^1 + 9t^0)^2 = 64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0;  
\ 7t^5 + 0t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0$.  

$+$: \ 7t^5 + 8t^4 + 9t^2 + 11t^1 + 5t^0; \ 7t^5 + 8t^4 + 9t^3 + 0t^2 + 1t^1 + 5t^0.$

Faster: multiply $a, b$ polynomials, square $c$ polynomial, add, carry.

$$(6t^4 + 23t^3 + 18t^2 + 29t^1 + 4t^0) + (64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0) = 70t^4 + 71t^3 + 171t^2 + 83t^1 + 85t^0;$$

$7t^5 + 8t^4 + 9t^3 + 0t^2 + 1t^1 + 5t^0$.  

Eliminate intermediate carries.

Outweighs cost of handling slightly larger coefficients.

Important to carry between multiplications (and squarings) to reduce coefficient size; but carries are usually a bad idea before additions, subtractions, etc.
Negative coeffs

Recall 159 → 15, 9.
Scaled: 15900 → 15000, 900.
Alternative: 159 → 16, −1.
Scaled: 15900 → 16000, −100.

Use digits \{-5, -4, \ldots, 4, 5\} instead of \{0, 1, \ldots, 9\}.

Some small advantages:
easily handle negative integers;
easily handle subtraction;
reduce products a bit.

Speedup: delay carries

Computing (e.g.) big ab + c^2:
multiply a, b polynomials, carry,
square c poly, carry, add, carry.

\[ (3t^2 + t^1 + 4t^0)(2t^2 + 7t^1 + 1t^0) = 6t^4 + 23t^3 + 18t^2 + 29t^1 + 4t^0; \]
carry: \[ 8t^4 + 5t^3 + 0t^2 + 9t^1 + 4t^0. \]

As before \[ (8t^2 + 3t^1 + 9t^0)^2 = 64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0; \]
\[ 7t^5 + 0t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0. \]

\[ +: 7t^5 + 8t^4 + 8t^3 + 9t^2 + 11t^1 + 5t^0; \]
\[ 7t^5 + 8t^4 + 9t^3 + 0t^2 + 1t^1 + 5t^0. \]

Faster: multiply a, b polynomials,
square c polynomial, add, carry.

\[ (6t^4 + 23t^3 + 18t^2 + 29t^1 + 4t^0) + (64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0) = 70t^4 + 71t^3 + 172t^2 + 83t^1 + 85t^0; \]
\[ 7t^5 + 8t^4 + 9t^3 + 0t^2 + 1t^1 + 5t^0. \]

Eliminate intermediate carries.
Outweighs cost of handling slightly larger coefficients.

Important to carry between multiplications (and squarings)
to reduce coefficient size; but carries are usually a bad idea
before additions, subtractions, etc.
Speedup: delay carries

Computing (e.g.) big $ab + c^2$:
multiply $a, b$ polynomials, carry,
square $c$ poly, carry, add, carry.

e.g. $a = 314, b = 271, c = 839$:
$(3t^2 + 1t^1 + 4t^0)(2t^2 + 7t^1 + 1t^0) = 6t^4 + 23t^3 + 18t^2 + 29t^1 + 4t^0$;
carry: $8t^4 + 5t^3 + 0t^2 + 9t^1 + 4t^0$.

As before $(8t^2 + 3t^1 + 9t^0)^2 = 64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0$;
$7t^5 + 0t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0$.

$+: 7t^5 + 8t^4 + 8t^3 + 9t^2 + 11t^1 + 5t^0$;
$7t^5 + 8t^4 + 9t^3 + 0t^2 + 1t^1 + 5t^0$.

Faster: multiply $a, b$ polynomials,
square $c$ polynomial, add, carry.

$(6t^4 + 23t^3 + 18t^2 + 29t^1 + 4t^0)$
$(64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0)$
$= 70t^4 + 71t^3 + 171t^2 + 83t^1 + 85t^0$;
$7t^5 + 8t^4 + 9t^3 + 0t^2 + 1t^1$.

Eliminate intermediate carries.
Outweighs cost of handling slightly larger coefficients.

Important to carry between multiplications (and squarings)
to reduce coefficient size;
but carries are usually a bad idea before additions, subtraction.

Digit {-5, -4, ..., 4, 5}
Speedup: delay carries
Computing (e.g.) big \( ab + c^2 \):
multiply \( a \), \( b \) polynomials, carry,
square \( c \) poly, carry, add, carry.

e.g. \( a = 314 \), \( b = 271 \), \( c = 839 \):
\[
(3t^2 + t^1 + 4t^0)(2t^2 + 7t^1 + 1t^0) = 6t^4 + 23t^3 + 18t^2 + 29t^1 + 4t^0;
\]
carry: \( 8t^4 + 5t^3 + 0t^2 + 9t^1 + 4t^0 \).

As before
\[
(8t^2 + 3t^1 + 9t^0)^2 = 64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0;
\]
\( 7t^5 + 0t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0 \).

\[+\]
\[
7t^5 + 8t^4 + 8t^3 + 9t^2 + 11t^1 + 5t^0;
\]
\( 7t^5 + 8t^4 + 9t^3 + 0t^2 + 1t^1 + 5t^0 \).

Faster: multiply \( a \), \( b \) polynomials, square \( c \) polynomial, add, carry.

\[
(6t^4 + 23t^3 + 18t^2 + 29t^1 + 4t^0) + (64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0) = 70t^4 + 71t^3 + 171t^2 + 83t^1 + 85t^0; \]

\( 7t^5 + 8t^4 + 9t^3 + 0t^2 + 1t^1 + 5t^0 \).

Eliminate intermediate carries. Outweighs cost of handling slightly larger coefficients.

Important to carry between multiplications (and squarings) to reduce coefficient size; but carries are usually a bad idea before additions, subtractions, etc.
Speedup: delay carries
Computing (e.g.) big \( ab + c^2 \):
\( a, b \) polynomials, carry, \( c \) poly, carry, add, carry.
E.g. \( a = 314, b = 271, c = 839 \):
\((314 + 4t^0)(2t^2 + 7t^1 + 1t^0) = 68t^3 + 18t^2 + 29t^1 + 4t^0; \)
\( t^4 + 5t^3 + 0t^2 + 9t^1 + 4t^0 \).
\((8t^2 + 3t^1 + 9t^0)^2 =
16t^4 + 23t^3 + 18t^2 + 29t^1 + 4t^0; \)
\( t^4 + 3t^3 + 9t^2 + 2t^1 + 1t^0 \).
\( 8t^4 + 8t^3 + 9t^2 + 11t^1 + 5t^0; \)
\( t^4 + 9t^3 + 0t^2 + 1t^1 + 5t^0 \).
Faster: multiply \( a, b \) polynomials, square \( c \) polynomial, add, carry.
\((6t^4 + 23t^3 + 18t^2 + 29t^1 + 4t^0) +
(64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0) =
70t^4 + 71t^3 + 171t^2 + 83t^1 + 85t^0; \)
\( 7t^5 + 8t^4 + 9t^3 + 0t^2 + 1t^1 + 5t^0 \).
Eliminate intermediate carries.
Outweighs cost of handling slightly larger coefficients.
Important to carry between multiplications (and squarings)
to reduce coefficient size;
but carries are usually a bad idea before additions, subtractions, etc.

Faster: multiply \( a, b \) polynomials,
square \( c \) polynomial, add, carry.
\((6t^4 + 23t^3 + 18t^2 + 29t^1 + 4t^0) +
(64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0) =
70t^4 + 71t^3 + 171t^2 + 83t^1 + 85t^0; \)
\( 7t^5 + 8t^4 + 9t^3 + 0t^2 + 1t^1 + 5t^0 \).
Eliminate intermediate carries.
Outweighs cost of handling slightly larger coefficients.
Important to carry between multiplications (and squarings)
to reduce coefficient size;
but carries are usually a bad idea before additions, subtractions, etc.

Using the obvious method:
\( 400 \) coeff mults, \( 361 \) coeff adds.
Faster: Write \( f \) as \( F_0 + F_1 t^{10} \);
\( g \) as \( G_0 + G_1 t^{10} \).
Then \( fg = (F_0 + F_1)(G_0 + G_1) t^{10} + (F_0G_0 - F_1G_1 t^{10})(1 - t^{10}) \).
Faster: multiply \(a, b\) polynomials, square \(c\) polynomial, add, carry.

\[
(6t^4 + 23t^3 + 18t^2 + 29t + 4) + (64t^4 + 48t^3 + 153t^2 + 54t + 81) = 70t^4 + 71t^3 + 171t^2 + 83t + 85.
\]

Eliminate intermediate carries. Outweighs cost of handling slightly larger coefficients.

Important to carry between multiplications (and squarings) to reduce coefficient size; but carries are usually a bad idea before additions, subtractions, etc.

**Speedup: polynomial Karatsuba**

How much work to multiply polys \(f = f_0 + f_1 t + \cdots + f_{19} t^{19}\), \(g = g_0 + g_1 t + \cdots + g_{19} t^{19}\)?

Using the obvious method: 400 coeff mults, 361 coeff adds.

Faster: Write \(f\) as \(F_0 + F_1 t^{10}\); \(F_0 = f_0 + f_1 t + \cdots + f_9 t^9\); \(F_1 = f_{10} + f_{11} t + \cdots + f_{19} t^9\).

Similarly write \(g\) as \(G_0 + G_1 t^{10}\).

Then \(fg = (F_0 + F_1 t^{10})(G_0 + G_1 t^{10}) + (F_0 G_0 - F_1 G_1 t^{10}) \cdot (1 - t^{10})\).
Faster: multiply $a, b$ polynomials, square $c$ polynomial, add, carry.

\[(6t^4 + 23t^3 + 18t^2 + 29t^1 + 4t^0) + (64t^4 + 48t^3 + 153t^2 + 54t^1 + 81t^0) = 70t^4 + 71t^3 + 171t^2 + 83t^1 + 85t^0;\]
\[7t^5 + 8t^4 + 9t^3 + 0t^2 + 1t^1 + 5t^0.\]

Eliminate intermediate carries. Outweighs cost of handling slightly larger coefficients.

Important to carry between multiplications (and squarings) to reduce coefficient size; but carries are usually a bad idea before additions, subtractions, etc.
Faster: multiply $a, b$ polynomials, square $c$ polynomial, add, carry.

$$(6t^4 + 23t^3 + 18t^2 + 29t + 4t^0) + (64t^4 + 48t^3 + 153t^2 + 54t + 81t^0) = 70t^4 + 71t^3 + 171t^2 + 83t + 85t^0;$$

$$7t^5 + 8t^4 + 9t^3 + 0t^2 + 1t + 5t^0.$$  

Eliminate intermediate carries. Outweighs cost of handling slightly larger coefficients.

Important to carry between multiplications (and squarings) to reduce coefficient size; but carries are usually a bad idea before additions, subtractions, etc.

---

Speedup: polynomial Karatsuba

How much work to multiply polys $f = f_0 + f_1 t + \cdots + f_{19} t^{19}$, $g = g_0 + g_1 t + \cdots + g_{19} t^{19}$?

Using the obvious method: 400 coeff mults, 361 coeff adds.

Faster: Write $f$ as $F_0 + F_1 t^{10}$; $F_0 = f_0 + f_1 t + \cdots + f_9 t^9$; $F_1 = f_{10} + f_{11} t + \cdots + f_{19} t^9$.

Similarly write $g$ as $G_0 + G_1 t^{10}$.

Then $fg = (F_0 + F_1)(G_0 + G_1)t^{10} + (F_0G_0 - F_1G_1 t^{10})(1 - t^{10})$.  

Faster: multiply polynomials, square polynomial, add, carry.

\[ (6t^4 + 23t^3 + 18t^2 + 29t + 40 + 48t^3 + 153t^2 + 54t + 81) + (71t^3 + 171t^2 + 83t + 85) = 70t^4 + 71t^3 + 171t^2 + 83t + 85; \]

\[ 7t^5 + 8t^4 + 9t^3 + 0t^2 + 1t + 5t. \]

Eliminate intermediate carries.
Outweighs cost of handling slightly larger coefficients.

Important to carry between multiplications (and squarings) to reduce coefficient size; 
but carries are usually a bad idea before additions, subtractions, etc.

Speedup: polynomial Karatsuba

How much work to multiply polys
\[ f = f_0 + f_1t + \cdots + f_{19}t^{19}, \]
\[ g = g_0 + g_1t + \cdots + g_{19}t^{19}? \]

Using the obvious method:
400 coeff mults, 361 coeff adds.

Faster: Write \( f \) as \( F_0 + F_1t^{10}; \)
\[ F_0 = f_0 + f_1t + \cdots + f_9t; \]
\[ F_1 = f_{10} + f_{11}t + \cdots + f_{19}t^9. \]

Similarly write \( g \) as \( G_0 + G_1t^{10}. \)

Then \( fg = (F_0 + F_1)(G_0 + G_1)t^{10} \]
\[ + (F_0G_0 - F_1G_1t^{10})(1 - t^{10}). \]

Larger coefficients, slight expense;
still saves time.
Can apply idea recursively as poly degree grows.
Faster: multiply $a; b$ polynomials, square $c$ polynomial, add, carry.

$$(6t^4 + 23t^3 + 18t^2 + 29t + 4) + (64t^4 + 48t^3 + 153t^2 + 54t + 81) = 70t^4 + 71t^3 + 171t^2 + 83t + 85;$$

$7t^5 + 8t^4 + 9t^3 + 0t^2 + 1t + 5t^0.$

Eliminate intermediate carries.

Outweighs cost of handling slightly larger coefficients.

Important to carry between multiplications (and squarings) to reduce coefficient size; but carries are usually a bad idea before additions, subtractions, etc.

Speedup: polynomial Karatsuba

How much work to multiply polys

$f = f_0 + f_1 t + \cdots + f_{19} t^{19},$

g = g_0 + g_1 t + \cdots + g_{19} t^{19}?$

Using the obvious method:

400 coeff mults, 361 coeff adds.

Faster: Write $f$ as $F_0 + F_1 t^{10};$

$F_0 = f_0 + f_1 t + \cdots + f_9 t^9;$

$F_1 = f_{10} + f_{11} t + \cdots + f_{19} t^9.$

Similarly write $g$ as $G_0 + G_1 t^{10}.$

Then $fg = (F_0 + F_1)(G_0 + G_1)t^{10}$

$+ (F_0G_0 - F_1G_1 t^{10})(1 - t^{10}).$

20 adds for $F_0 + H_1,$

300 mults for three products $F_0G_0, F_1G_1, (F_0 + F_1)(G_0 + G_1)$

243 adds for those products.

9 adds for $F_0G_0 - F_1G_1 t^{10}$ with subs counted as adds and with delayed negations.

19 adds for $\cdots (1 - t^{10}).$

19 adds to finish.

Total 300 mults, 310 adds.

Larger coefficients, slight expense; still saves time.

Can apply idea recursively as poly degree grows.
Speedup: polynomial Karatsuba

How much work to multiply polys
\[ f = f_0 + f_1 t + \cdots + f_{19} t^{19}, \]
\[ g = g_0 + g_1 t + \cdots + g_{19} t^{19}? \]

Using the obvious method:
400 coeff mults, 361 coeff adds.

Faster: Write \( f \) as \( F_0 + F_1 t^{10} \);
\[ F_0 = f_0 + f_1 t + \cdots + f_9 t^9; \]
\[ F_1 = f_{10} + f_{11} t + \cdots + f_{19} t^9. \]

Similarly write \( g \) as \( G_0 + G_1 t^{10} \).

Then \( fg = (F_0 + F_1)(G_0 + G_1)t^{10} + (F_0G_0 - F_1G_1 t^{10})(1 - t^{10}). \)

20 adds for \( F_0 + F_1, G_0 + G_1 \).
300 mults for three products \( F_0G_0, F_1G_1, (F_0 + F_1)(G_0 + G_1) \).
243 adds for those products.
9 adds for \( F_0G_0 - F_1G_1 t^{10} \) with subs counted as adds and with delayed negations.
19 adds for \( (1 - t^{10}) \).
19 adds to finish.

Total 300 mults, 310 adds.
Larger coefficients, slight expense; still saves time.
Can apply idea recursively as poly degree grows.
Speedup: polynomial Karatsuba

How much work to multiply polys

\[ f = f_0 + f_1 t + \cdots + f_{19} t^{19}, \]
\[ g = g_0 + g_1 t + \cdots + g_{19} t^{19}. \]

Using the obvious method:

400 coeff mults, 361 coeff adds.

Faster: Write \( f \) as \( F_0 + F_1 t^{10}; \)
\[ F_0 = f_0 + f_1 t + \cdots + f_9 t^9; \]
\[ F_1 = f_{10} + f_{11} t + \cdots + f_{19} t^9. \]

Similarly write \( g \) as \( G_0 + G_1 t^{10}. \)

Then \( fg = (F_0 + F_1)(G_0 + G_1)t^{10} + (F_0G_0 - F_1G_1 t^{10})(1 - t^{10}). \)

20 adds for \( F_0 + F_1, G_0 + G_1. \)
300 mults for three products \( F_0G_0, F_1G_1, (F_0 + F_1)(G_0 + G_1). \)
243 adds for those products.
9 adds for \( F_0G_0 - F_1G_1 t^{10} \)
with subs counted as adds and with delayed negations.
19 adds for \( \cdots (1 - t^{10}). \)
19 adds to finish.

Total 300 mults, 310 adds.
Larger coefficients, slight expense; still saves time.

Can apply idea recursively as poly degree grows.
Speedup: polynomial Karatsuba
How much work to multiply polys
\[ f = f_0 + f_1 t + \cdots + f_{19} t^{19}, \]
\[ g = g_0 + g_1 t + \cdots + g_{19} t^{19}. \]

Using the obvious method:
- 400 coeff mults, 361 coeff adds.

Faster: Write
\[ f = F_0 + F_1 t^{10}; \]
\[ + f_1 t + \cdots + f_9 t^9; \]
\[ + f_{11} t + \cdots + f_{19} t^9. \]

Write \( g \) as \( G_0 + G_1 t^{10}. \)
\[ g = (F_0 + F_1)(G_0 + G_1) t^{10} \]
\[ - F_1 G_1 t^{10})(1 - t^{10}). \]

20 adds for \( F_0 + F_1, G_0 + G_1. \)
300 mults for three products \( F_0 G_0, F_1 G_1, (F_0 + F_1)(G_0 + G_1). \)
243 adds for those products.
9 adds for \( F_0 G_0 - F_1 G_1 t^{10} \)
with subs counted as adds and with delayed negations.
19 adds for \( \cdots (1 - t^{10}). \)
19 adds to finish.

Total 300 mults, 310 adds.
Larger coefficients, slight expense;
still saves time.
Can apply idea recursively as poly degree grows.

Many other algebraic speedups in polynomial multiplication:
"Toom," "FFT," etc.
Increasingly important as polynomial degree grows.
\[ O(n \lg n \lg \lg n) \]
coeff operations to compute \( n \)-coeff product.
Useful for sizes of \( n \) that occur in cryptography?
In some cases, yes!
But Karatsuba is the limit for prime-field ECC/ECDLP on most current CPUs.
Speedup: polynomial Karatsuba

To multiply polys
\[ f = f_0 + f_1 t + \cdots + f_{19} t^{19}, \]
\[ g = g_0 + g_1 t + \cdots + g_{19} t^{19}. \]

Using the obvious method:
200 coeff mults, 181 coeff adds.

Faster: Write \( f \) as \( F_0 + F_1 t^{10} \);
24 adds for \( F_0 + F_1 \), \( G_0 + G_1 \).
300 mults for three products \( F_0 G_0, \ F_1 G_1, (F_0 + F_1)(G_0 + G_1) \).
243 adds for those products.
9 adds for \( F_0 G_0 - F_1 G_1 t^{10} \)
with subs counted as adds
and with delayed negations.
19 adds for \( \cdots (1 - t^{10}) \).
19 adds to finish.

Total 300 mults, 310 adds.

Larger coefficients, slight expense;
still saves time.

Can apply idea recursively
as poly degree grows.

Many other algebraic speedups
in polynomial multiplication:
“Toom,” “FFT,” etc.

Increasingly important as
polynomial degree grows.
\( O(n \log n \log \log n) \) coeff operations
to compute \( n \)-coeff product.

Useful for sizes of \( n \) that occur in cryptography?
In some cases, yes!
But Karatsuba is the limit
for prime-field ECC/ECDLP
on most current CPUs.
Speedup: polynomial Karatsuba
How much work to multiply polys
\[ f = f_0 + f_1 t + \cdots + f_{19} t^{19}, \]
\[ g = g_0 + g_1 t^{10} + \cdots + g_{19} t^{19}. \]

Using the obvious method:

400 coeff mults, 361 coeff adds.

Faster: Write
\[ f = F_0 + F_1 t^{10} + \cdots + F_{19} t^{19}, \]
\[ g = G_0 + G_1 t^{10} + \cdots + G_{19} t^{19}. \]

Then
\[ f g = (F_0 + F_1)(G_0 + G_1) t^{10} + (F_0 G_0 - F_1 G_1 t^{10})(1 - t^{10}). \]

13

20 adds for \( F_0 + F_1, G_0 + G_1 \).
300 mults for three products \( F_0 G_0, F_1 G_1, (F_0 + F_1)(G_0 + G_1) \).
243 adds for those products.
9 adds for \( F_0 G_0 - F_1 G_1 t^{10} \)
with subs counted as adds
and with delayed negations.
19 adds for \( \cdots (1 - t^{10}) \).
19 adds to finish.

Total 300 mults, 310 adds.

Larger coefficients, slight expense; still saves time.

Can apply idea recursively as poly degree grows.

14

Many other algebraic speedups in polynomial multiplication:

“Toom,” “FFT,” etc.

Increasingly important as polynomial degree grows.

\[ O(n \lg n \lg \lg n) \] coeff operations to compute \( n \)-coeff product.

Useful for sizes of \( n \) that occur in cryptography?
In some cases, yes!

But Karatsuba is the limit for prime-field ECC/ECDLP on most current CPUs.
20 adds for $F_0 + F_1, G_0 + G_1$.
300 mults for three products
$F_0 G_0, F_1 G_1, (F_0 + F_1)(G_0 + G_1)$.
243 adds for those products.
9 adds for $F_0 G_0 - F_1 G_1 t^{10}$
with subs counted as adds
and with delayed negations.
19 adds for $\cdots (1 - t^{10})$.
19 adds to finish.

Total 300 mults, 310 adds.
Larger coefficients, slight expense;
still saves time.

Can apply idea recursively
as poly degree grows.

Many other algebraic speedups
in polynomial multiplication:
“Toom,” “FFT,” etc.

Increasingly important as
polynomial degree grows.
$O(n \log n \log \log n)$ coeff operations

to compute $n$-coeff product.

Useful for sizes of $n$
that occur in cryptography?
In some cases, yes!
But Karatsuba is the limit
for prime-field ECC/ECDLP
on most current CPUs.
for $F_0 + F_1, G_0 + G_1$.

300 mults for three products
$(F_0 + F_1)(G_0 + G_1)$.

243 adds for those products.

9 adds for $F_0 G_0 - F_1 G_1 t^{10}$
was counted as adds
in delayed negations.

19 adds for $\cdots (1 - t^{10})$.

19 adds to finish.

Total 300 mults, 310 adds.
Larger coefficients, slight expense;
still saves time.
Can apply idea recursively
as poly degree grows.

Many other algebraic speedups
in polynomial multiplication:
“Toom,” “FFT,” etc.

Increasingly important as polynomial degree grows.
$O(n \log n \log \log n)$ coeff operations
to compute $n$-coeff product.

Useful for sizes of $n$
that occur in cryptography?
In some cases, yes!
But Karatsuba is the limit
for prime-field ECC/ECDLP
on most current CPUs.

Modular reduction
How to compute $f \mod p$?
Can use definition:
$f \mod p = f - p \lfloor f/p \rfloor$.
Can multiply $f$ by a
precomputed $1 = p$ approximation;
easily adjust to obtain
$\lfloor f/p \rfloor$.
Slight speedup: “2-adic inverse”;
“Montgomery reduction.”
Many other algebraic speedups in polynomial multiplication: “Toom,” “FFT,” etc.

Increasingly important as polynomial degree grows. $O(n \lg n \lg \lg n)$ coeff operations to compute $n$-coeff product.

Useful for sizes of $n$ that occur in cryptography? In some cases, yes!

But Karatsuba is the limit for prime-field ECC/ECDLP on most current CPUs.

Modular reduction

How to compute $f \mod p$?

Can use definition: $f \mod p = f - p \lfloor f/p \rfloor$.

Can multiply $f$ by a precomputed $1/p$ approximation; easily adjust to obtain $\lfloor f/p \rfloor$.

Slight speedup: “2-adic inverse”; “Montgomery reduction.”
Many other algebraic speedups in polynomial multiplication: “Toom,” “FFT,” etc.

Increasingly important as polynomial degree grows. $O(n \lg n \lg \lg n)$ coeff operations to compute $n$-coeff product.

Useful for sizes of $n$ that occur in cryptography? In some cases, yes!
But Karatsuba is the limit for prime-field ECC/ECDLP on most current CPUs.

Modular reduction

How to compute $f \mod p$?

Can use definition: $f \mod p = f - p \lfloor f/p \rfloor$.

Can multiply $f$ by a precomputed $1/p$ approximation; easily adjust to obtain $\lfloor f/p \rfloor$.

Slight speedup: “2-adic inverse”; “Montgomery reduction.”
Many other algebraic speedups in polynomial multiplication: “Toom,” “FFT,” etc.

Increasingly important as polynomial degree grows. \(O(n \lg n \lg \lg n)\) coeff operations to compute \(n\)-coeff product.

Useful for sizes of \(n\) that occur in cryptography? In some cases, yes!

But Karatsuba is the limit for prime-field ECC/ECDLP on most current CPUs.

Modular reduction

How to compute \(f \mod p\)?

Can use definition:
\[f \mod p = f - p \lfloor f/p \rfloor.\]

Can multiply \(f\) by a precomputed \(1/p\) approximation; easily adjust to obtain \(\lfloor f/p \rfloor\).

Slight speedup: “2-adic inverse”; “Montgomery reduction.”
Many other algebraic speedups in polynomial multiplication: “Toom,” “FFT,” etc. Increasingly important as polynomial degree grows. \((O(n \lg \lg n))\) coeff operations to compute \(n\)-coeff product.

For sizes of \(n\) that occur in cryptography? In some cases, yes! But Karatsuba is the limit for prime-field ECC/ECDLP on most current CPUs.

**Modular reduction**

How to compute \(f \mod p\)? Can use definition: \(f \mod p = f - p \lfloor \frac{f}{p} \rfloor\). Can multiply \(f\) by a precomputed \(1/p\) approximation; easily adjust to obtain \(\lfloor \frac{f}{p} \rfloor\).

Slight speedup: “2-adic inverse”; “Montgomery reduction.”

e.g. \(314159265358 \mod 271828\):
Precompute \(\lfloor \frac{1000000000000}{271828} \rfloor = 3678796\).

Compute \(314159 \cdot 3678796\) = ...
Compute \(314159265358 - 1155726 \cdot 271828\) = 578230.
Oops, too big:
\(578230 - 271828 = 306402\).
\(306402 - 271828 = 34574\).
Many other algebraic speedups in polynomial multiplication: “Toom,” “FFT,” etc.
Increasingly important as polynomial degree grows.
\(O(n \log n \log \log n)\) coeff operations to compute \(n\)-coeff product.
Useful for sizes of \(n\) that occur in cryptography?
In some cases, yes!
But Karatsuba is the limit for prime-field ECC/ECDLP on most current CPUs.

---

**Modular reduction**

How to compute \(f \mod p\)?

Can use definition:
\(f \mod p = f - p \lfloor f/p \rfloor\).
Can multiply \(f\) by a precomputed \(1/p\) approximation; easily adjust to obtain \(\lfloor f/p \rfloor\).

Slight speedup: “2-adic inverse”; “Montgomery reduction.”

---

e.g. \(314159265358 \mod 271828\):

Precompute \(\lfloor 1000000000000/271828 \rfloor = 3678796\).

Compute \(314159 \cdot 3678796 = 1155726872564\).

Compute \(314159265358 - 1155726872564 = 578230\).
Oops, too big:
\(578230 - 271828 = 306402\).
\(306402 - 271828 = 34574\).
Many other algebraic speedups in polynomial multiplication: "Toom," "FFT," etc.
Increasingly important as polynomial degree grows.

$O(n \log n \log \log n)$ coeff operations to compute $n$-coeff product.

Useful for sizes of $n$ that occur in cryptography?
In some cases, yes!
But Karatsuba is the limit for prime-field ECC/ECDLP on most current CPUs.

Modular reduction
How to compute $f \mod p$?
Can use definition:
$f \mod p = f - p \lfloor f/p \rfloor$.
Can multiply $f$ by a precomputed $1/p$ approximation;
easily adjust to obtain $\lfloor f/p \rfloor$.

Slight speedup: “2-adic inverse”; “Montgomery reduction.”

E.g. $314159265358 \mod 271828$:
Precompute
$\lfloor 1000000000000/271828 \rfloor$
$= 3678796$.
Compute
$314159 \cdot 3678796$
$= 1155726872564$.
Compute
$314159265358 - 1155726 \cdot 271828$
$= 578230$.
Oops, too big:
$578230 - 271828 = 306402$.
$306402 - 271828 = 34574$. 
Modular reduction

How to compute $f \mod p$?

Can use definition:

$$f \mod p = f - p \lfloor f/p \rfloor.$$ 

Can multiply $f$ by a precomputed $1/p$ approximation; easily adjust to obtain $\lfloor f/p \rfloor$.

Slight speedup: “2-adic inverse”; “Montgomery reduction.”

e.g. $314159265358 \mod 271828$:

Precompute

$$\lfloor 1000000000000/271828 \rfloor = 3678796.$$ 

Compute

$$314159 \cdot 3678796 = 1155726872564.$$ 

Compute

$$314159265358 - 1155726 \cdot 271828 = 578230.$$ 

Oops, too big:

$$578230 - 271828 = 306402.$$ 

$$306402 - 271828 = 34574.$$
Modular reduction

How to compute \( f \mod p \)?

Can use definition:
\[
 f \mod p = f - p \lfloor f/p \rfloor.
\]

Multiply \( f \) by a precomputed \( 1/p \) approximation; easily adjust to obtain \( \lfloor f/p \rfloor \).

Slight speedup: “2-adic inverse”; “Montgomery reduction.”

e.g. \( 314159265358 \mod 271828 \):

Precompute
\[
\lfloor 1000000000000/271828 \rfloor
= 3678796.
\]

Compute
\[
314159 \cdot 3678796
= 1155726872564.
\]

Compute
\[
314159265358 - 1155726 \cdot 271828
= 578230.
\]

Oops, too big:
\[
578230 - 271828 = 306402.
\]
\[
306402 - 271828 = 34574.
\]

We can do better: normally \( p \) is chosen with a special form to make \( f \mod p \) much faster.

Special primes hurt security for \( \mathbf{F}^*_p \), \( \text{Clock}(\mathbf{F}_p) \), etc., but not for elliptic curves!

Curve25519:
\[
p = 2^{255} - 19.
\]

NIST P-224:
\[
p = 2^{224} - 2^{96} + 1.
\]

secp112r1:
\[
p = (2^{128} - 3) = 76439.
\]

Divides special form.

gls1271: \( p = 2^{127} - 1 \), with degree-2 extension (a bit scary).
Modular reduction

How to compute $f \mod p$?

Can use definition:

$$f \mod p = f - p \lfloor f / p \rfloor.$$

Can multiply $f$ by a precomputed $1/p$ approximation; easily adjust to obtain $\lfloor f / p \rfloor$.

Slight speedup: "2-adic inverse"; "Montgomery reduction."

We can do better: normally $p$ is chosen with a special form to make $f \mod p$ much faster.

Special primes hurt security for $\mathbb{F}_p^*$, Clock($\mathbb{F}_p$), etc., but not for elliptic curves!

Curve25519: $p = 2^{255} - 19$.

NIST P-224: $p = 2^{224} - 2^{96} + 1$.

secp112r1: $p = (2^{112} - 3)$.

Divides special form.

gls1271: $p = 2^{127} - 1$, with degree-2 extension (a bit scary).
16

How to compute \( f \mod p \)?

Can use definition:

\[
 f \mod p = f - p \left\lfloor \frac{f}{p} \right\rfloor .
\]

Can multiply \( f \) by a precomputed \( \frac{1}{p} \) approximation; easily adjust to obtain \( \left\lfloor \frac{f}{p} \right\rfloor \).

Slight speedup: "2-adic inverse"; "Montgomery reduction."

e.g. 314159265358 mod 271828:

Precompute

\[
\left\lfloor \frac{1000000000000}{271828} \right\rfloor = 3678796.
\]

Compute

\[
314159 \cdot 3678796 = 1155726872564.
\]

Compute

\[
314159265358 - 1155726 \cdot 271828 = 578230.
\]

Oops, too big:

\[
578230 - 271828 = 306402.
\]

\[
306402 - 271828 = 34574.
\]

17

We can do better: normally \( p \) is chosen with a special form to make \( f \mod p \) much faster.

Special primes hurt security for \( F^*, \text{Clock}(F_p) \), etc., but not for elliptic curves!

Curve25519: \( p = 2^{255} - 19 \).

NIST P-224: \( p = 2^{224} - 2^{96} + 1 \).

secp112r1: \( p = (2^{128} - 3)/7 \).

Divides special form.

gls1271: \( p = 2^{127} - 1 \), with degree-2 extension (a bit scary).
e.g. 314159265358 mod 271828:
Precompute
\[
\left\lfloor \frac{10000000000000}{271828} \right\rfloor = 3678796.
\]
Compute
\[
314159 \cdot 3678796 = 1155726872564.
\]
Compute
\[
314159265358 - 1155726 \cdot 271828 = 578230.
\]
Oops, too big:
\[
578230 - 271828 = 306402.
\]
\[
306402 - 271828 = 34574.
\]

We can do better: normally \( p \) is chosen with a special form to make \( f \mod p \) much faster.

Special primes hurt security for \( F_p^* \), \( \text{Clock}(F_p) \), etc., but not for elliptic curves!

Curve25519: \( p = 2^{255} - 19 \).

NIST P-224: \( p = 2^{224} - 2^{96} + 1 \).

secp112r1: \( p = (2^{128} - 3)/76439 \).

Divides special form.

gls1271: \( p = 2^{127} - 1 \), with degree-2 extension (a bit scary).
e.g. 314159265358 mod 271828:
Precompute
⌊1000000000000/271828⌋ = 3678796.
Compute
314159 · 3678796 = 1155726872564.
Compute
314159265358 − 1155726 · 271828 = 578230.
Oops, too big:
578230 − 271828 = 306402.
306402 − 271828 = 34574.

We can do better: normally
$p$ is chosen with a special form
to make $f$ mod $p$ much faster.
Special primes hurt security
for $\mathbb{F}_p^*$, $\text{Clock}(\mathbb{F}_p)$, etc.,
but not for elliptic curves!

Curve25519: $p = 2^{255} - 19$.
NIST P-224: $p = 2^{224} - 2^{96} + 1$.
secp112r1: $p = (2^{128} - 3)/76439$.

Divides special form.
gls1271: $p = 2^{127} - 1$, with
degree-2 extension (a bit scary).

Small example: $p = 1000003$.
Then $1000000 a + b \equiv b - 3 a$.
e.g. 314159265358 = 314159 · 1000000 + 265358
$-942477 + 265358 = -677119$.
Easily adjust $b - 3 a$
to the range $\{ 0; 1; \ldots; p - 1 \}$
by adding/subtracting a few $p$'s:
e.g. $-677119 \equiv 322884$. 
We can do better: normally $p$ is chosen with a special form to make $f \mod p$ much faster.

Special primes hurt security for $\mathbf{F}_p^*$, $\text{Clock}(\mathbf{F}_p)$, etc., but not for elliptic curves!

Curve25519: $p = 2^{255} - 19$.

NIST P-224: $p = 2^{224} - 2^{96} + 1$.

secp112r1: $p = (2^{128} - 3)/76439$.

Divides special form.

gls1271: $p = 2^{127} - 1$, with degree-2 extension (a bit scary).

Small example: $p = 1000003$.

Then $1000000a + b \equiv b - 3a$.

e.g. $314159265358 = 314159 \cdot 1000000 + 265358 \equiv 314159(-3) + 265358 = -942477 + 265358 = -677119$.

Easily adjust $b - 3a$ to the range $\{0, 1, \ldots, p - 1\}$ by adding/subtracting a few $p$'s:

e.g. $-677119 \equiv 322884$. 

We can do better: normally \( p \) is chosen with a special form to make \( f \) mod \( p \) much faster.

Special primes hurt security for \( \mathbb{F}_p^* \), \( \text{Clock}(\mathbb{F}_p) \), etc., but not for elliptic curves!

Curve25519: \( p = 2^{255} - 19 \).

NIST P-224: \( p = 2^{224} - 2^{96} + 1 \).

secp112r1: \( p = (2^{128} - 3)/76439 \). Divides special form.

gls1271: \( p = 2^{127} - 1 \), with degree-2 extension (a bit scary).

Small example: \( p = 1000003 \). Then \( 1000000a + b \equiv b - 3a \mod p \).

e.g. \( 314159265358 = 314159 \cdot 1000000 + 265358 \)
\( 314159(-3) + 265358 = -942477 + 265358 = -677119 \).

Easily adjust \( b - 3a \) to the range \( \{0, 1, \ldots, p-1\} \) by adding/subtracting a few \( p \)’s:

e.g. \(-677119 \equiv 322884\).
We can do better: normally \( p \) is chosen with a special form to make \( f \mod p \) much faster.

Special primes hurt security for \( \mathbf{F}_p^* \), \( \text{Clock}(\mathbf{F}_p) \), etc., but not for elliptic curves!

Curve25519: \( p = 2^{255} - 19 \).

NIST P-224: \( p = 2^{224} - 2^{96} + 1 \).

secp112r1: \( p = (2^{128} - 3)/76439 \).

Divides special form.

gls1271: \( p = 2^{127} - 1 \), with degree-2 extension (a bit scary).

Small example: \( p = 1000003 \).

Then \( 1000000a + b \equiv b - 3a \).

\[
\text{e.g. } 314159265358 = 314159 \cdot 1000000 + 265358 \equiv 314159(-3) + 265358 = -942477 + 265358 = -677119.
\]

Easily adjust \( b - 3a \) to the range \( \{0, 1, \ldots, p - 1\} \) by adding/subtracting a few \( p \)’s: e.g. \( -677119 \equiv 322884 \).
We can do better: normally \( p \) is chosen with a special form to make \( f \mod p \) much faster.

Special primes hurt security for \( \mathbb{F}_p^* \), \( \text{Clock}(\mathbb{F}_p) \), etc., but not for elliptic curves!

**Examples:**
- Curve25519: \( p = 2^{255} - 19 \).
- NIST P-224: \( p = 2^{224} - 2^{96} + 1 \).
- secp112r1: \( p = (2^{128} - 3) / 76439 \).
- Divides special form.
- \( \text{gls1271}: p = 2^{127} - 1 \), with degree-2 extension (a bit scary).

Small example: \( p = 1000003 \).
Then \( 1000000a + b \equiv b - 3a \).

\[
\begin{align*}
e.g.\ 314159265358 & = 314159 \cdot 1000000 + 265358 \equiv 314159(-3) + 265358 = \\
& = -942477 + 265358 = -677119.
\end{align*}
\]

Easily adjust \( b - 3a \) to the range \( \{0, 1, \ldots, p - 1\} \) by adding/subtracting a few \( p \)’s:

\[
\begin{align*}
e.g.\ -677119 & \equiv 322884.
\end{align*}
\]

Hmmm, is adjustment so easy?
Conditional branches are slow and leak secrets through timing.
Can eliminate the branches, but adjustment isn’t free.

Speedup: Skip the adjustment for intermediate results.

“Lazy reduction.”

Adjust only for output.

\( b - 3a \) is small enough to continue computations.
We can do better: normally $p$ is chosen with a special form to make $f \mod p$ much faster.

Special primes hurt security for $F^*_p$, $\text{Clock}(F_p)$, etc., but not for elliptic curves!

**Curve25519:** $p = 2^{255} - 19$.

**NIST P-224:** $p = 2^{224} - 2^{96} + 1$.

**secp112r1:** $p = (2^{128} - 3) = 76439$.

Divides special form.

**gls1271:** $p = 2^{127} - 1$, with degree-2 extension (a bit scary).

---

Small example: $p = 1000003$.

Then $1000000a + b \equiv b - 3a$.

e.g. $314159265358 = 314159 \cdot 1000000 + 265358 \equiv 314159(-3) + 265358 = -942477 + 265358 = -677119$.

Easily adjust $b - 3a$ to the range $\{0, 1, \ldots, p - 1\}$ by adding/subtracting a few $p$’s:

e.g. $-677119 \equiv 322884$.

---

Hmmm, is adjustment so easy?

Conditional branches are slow and leak secrets through timing.

Can eliminate the branches, but adjustment isn’t free.

**Speedup:** Skip the adjustment for intermediate results.

“Lazy reduction.”

Adjust only for output.

$b - 3a$ is small enough to continue computations.
We can do better: normally $p$ is chosen with a special form to make $f \mod p$ much faster.

Special primes hurt security for $F^\ast p$, $\text{Clock}(F^p)$, etc., but not for elliptic curves!

**Curve25519:**
$p = 2^{255} - 19$.

**NIST P-224:**
$p = 2^{224} - 2^96 + 1$.

**secp112r1:**
$p = (2^{128} - 3) = 76439$.

Divides special form.

**gls1271:**
$p = 2^{127} - 1$, with degree-2 extension (a bit scary).

---

Small example: $p = 1000003$.
Then $1000000a + b \equiv b - 3a$.

E.g. $314159265358 = 314159 \cdot 1000000 + 265358 \equiv 314159(-3) + 265358 = -942477 + 265358 = -677119$.

Easily adjust $b - 3a$ to the range $\{0, 1, \ldots, p - 1\}$ by adding/subtracting a few $p$'s:

E.g. $-677119 \equiv 322884$.

---

Hmmm, is adjustment so easy?
Conditional branches are slow and leak secrets through timing.
Can eliminate the branches, but adjustment isn’t free.

Speedup: Skip the adjustment for intermediate results.
“Lazy reduction.”
Adjust only for output.

$b - 3a$ is small enough to continue computations.
Small example: $p = 1000003$. Then $1000000a + b \equiv b - 3a$.

E.g. $314159265358 = 314159 \cdot 1000000 + 265358 \equiv 314159(-3) + 265358 = -942477 + 265358 = -677119$.

Easily adjust $b - 3a$ to the range $\{0, 1, \ldots, p - 1\}$ by adding/subtracting a few $p$’s: e.g. $-677119 \equiv 322884$.

Hmmm, is adjustment so easy? Conditional branches are slow and leak secrets through timing. Can eliminate the branches, but adjustment isn’t free.

Speedup: Skip the adjustment for intermediate results. “Lazy reduction.” Adjust only for output.

$b - 3a$ is small enough to continue computations.
19
Small example: \( p = 1000003 \).
Then \( 1000000 a + b \equiv b - 3a \).
e.g. \( 314159265358 = 314159 \cdot 1000000 + 265358 \equiv 314159(-3) + 265358 = 7 + 265358 = 266076 \).

Adjust \( b - 3a \) to the range \( \{0, 1, \ldots, p - 1\} \) by adding/subtracting a few \( p \)’s: \( 266076 \equiv 321974 \).

Hmmm, is adjustment so easy? Conditional branches are slow and leak secrets through timing. Can eliminate the branches, but adjustment isn’t free.

Speedup: Skip the adjustment for intermediate results. “Lazy reduction.”
Adjust only for output.
\( b - 3a \) is small enough to continue computations.

20
Can delay carries until after multiplication.
e.g. To square 314159 in \( \mathbb{Z}/1000003 \): Square poly \( 3 t^5 + 1 t^4 + 4 t^3 + 1 t^2 + 5 t + 9 \), obtaining, \( 9 t^10 + 6 t^9 + 25 t^8 + 14 t^7 + 48 t^6 + 72 t^5 + 59 t^4 + 82 t^3 + 43 t^2 + 90 t + 81 \).

Reduce: replace \( (c_i) t^6+i \) by \( (-3c_i) t^i \), obtaining \( 64 t^3 - 32 t^2 + 48 t - 63 \).

Carry: \( 8 t^6 - 4 t^5 - 2 t^4 + t^3 + 2 t^2 + 2 t - 3 \).
Small example: \( p = 1000003 \).
Then \( 1000000a + b \equiv b - 3a \).

\[ 314159265358 = 314159 \cdot 1000000 + 265358 \equiv 314159(-3) + 265358 = -942477 + 265358 = -677119. \]

Easily adjust \( b - 3a \) to the range \( \{0; 1; \ldots; p - 1\} \) by adding/subtracting a few \( p \)'s:

\[ -677119 \equiv 322884. \]

Hmmm, is adjustment so easy? Conditional branches are slow and leak secrets through timing. Can eliminate the branches, but adjustment isn’t free.

Speedup: Skip the adjustment for intermediate results.

“Lazy reduction.”

Adjust only for output.

\( b - 3a \) is small enough to continue computations.

Can delay carries until after multiplication by 3.

\[ \text{e.g. To square 314159 in } \mathbb{Z}/1000003: \text{ Square poly} \]

\[ 3t^5 + 1t^4 + 4t^3 + 1t^2 + 5t + 9, \]

obtaining \[ 9t^{10} + 6t^9 + 14t^7 + 48t^6 + 72t^5 + 82t^3 + 43t^2 + 90t \].

Reduce: replace \((c_i)t^i\) by \((-3c_i)t^i\), obtaining \[ 64t^3 - 32t^2 + 48t \].

Carry: \[ 8t^6 - 4t^5 - t^3 + 2t^2 + 2t^1 - 3t \].
19

Small example: \( p = 1000003 \).

Then \( 1000000a + b \equiv b - 3a \).

e.g. \( 314159265358 = 314159 \cdot 1000000 + 265358 \equiv 314159(-3) + 265358 = -942477 + 265358 = -677119 \).

Easily adjust \( b - 3a \) to the range \( \{0; 1; \ldots; p-1\} \) by adding/subtracting a few \( p \)'s:

e.g. \( -677119 \equiv 322884 \).

20

Hmmm, is adjustment so easy?

Conditional branches are slow and leak secrets through timing.

Can eliminate the branches, but adjustment isn’t free.

Speedup: Skip the adjustment for intermediate results.

“Lazy reduction.”

Adjust only for output.

\( b - 3a \) is small enough to continue computations.

Can delay carries until after multiplication by 3.

e.g. To square 314159 in \( \mathbb{Z}/1000003 \): Square poly \( 3t^5 + 1t^4 + 4t^3 + 1t^2 + 5t^1 \)

obtaining \( 9t^{10} + 6t^9 + 25t^8 + 14t^7 + 48t^6 + 72t^5 + 59t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0 \).

Reduce: replace \((c_i)t^{6+i} \) by \((-3c_i)t^i \), obtaining \( 72t^5 + 3 \cdot 64t^3 - 32t^2 + 48t^1 - 63t^0 \).

Carry: \( 8t^6 - 4t^5 - 2t^4 + 1t^3 + 2t^2 + 2t^1 - 3t^0 \).
Hmmm, is adjustment so easy?
Conditional branches are slow and leak secrets through timing.
Can eliminate the branches, but adjustment isn’t free.

Speedup: Skip the adjustment for intermediate results.
“Lazy reduction.”
Adjust only for output.

\[ b - 3a \] is small enough to continue computations.

Can delay carries until after multiplication by 3.
e.g. To square 314159 in \( \mathbb{Z}/1000003 \): Square poly
\[ 3t^5 + 1t^4 + 4t^3 + 1t^2 + 5t^1 + 9t^0, \]

obtaining \[ 9t^{10} + 6t^9 + 25t^8 + 14t^7 + 48t^6 + 72t^5 + 59t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0. \]

Reduce: replace \((c_i)t^{6+i}\) by \((-3c_i)t^i\), obtaining

\[ 72t^5 + 32t^4 + 64t^3 - 32t^2 + 48t^1 - 63t^0. \]

Carry: \[ 8t^6 - 4t^5 - 2t^4 + 1t^3 + 2t^2 + 2t^1 - 3t^0. \]
Hmmm, is adjustment so easy?

Conditional branches are slow and leak secrets through timing. Can eliminate the branches, but adjustment isn't free.

Speedup: Skip the adjustment for intermediate results.

"Lazy reduction."

Adjust only for output.

$b - 3a$ is small enough to continue computations.

Can delay carries until after multiplication by 3.

e.g. To square 314159 in $\mathbb{Z}/1000003$: Square poly

$$3t^5 + t^4 + 4t^3 + t^2 + 5t^1 + 9t^0,$$

obtaining $9t^{10} + 6t^9 + 25t^8 + 14t^7 + 48t^6 + 72t^5 + 59t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0$.

Reduce: replace $(c_i)t^{6+i}$ by $(-3c_i)t^i$, obtaining $72t^5 + 32t^4 + 64t^3 - 32t^2 + 48t^1 - 63t^0$.

Finish reduction:

$$-5t^5 + 2t^4 + 64t^3 - 32t^2 + 48t^1 - 63t^0.$$
Hmmm, is adjustment so easy?
Conditional branches are slow
and leak secrets through timing.
Can eliminate the branches,
but adjustment isn’t free.
Speedup: Skip the adjustment
for intermediate results.
“Lazy reduction.”
Adjust only for output.

\[ b - 3a \text{ is small enough} \]

to continue computations.

\[ \text{Can delay carries until after} \]
\[ \text{multiplication by 3.} \]
\[ \text{e.g. To square } 314159 \]
\[ \text{in } \mathbb{Z}/1000003: \text{ Square poly} \]
\[ 3t^5 + 1t^4 + 4t^3 + 1t^2 + 5t^1 + 9t^0, \]
\[ \text{obtaining } 9t^{10} + 6t^9 + 25t^8 + \]
\[ 14t^7 + 48t^6 + 72t^5 + 59t^4 + \]
\[ 82t^3 + 43t^2 + 90t^1 + 81t^0. \]

Reduce: replace \((c_i)t^{6+i}\) by
\((-3c_i)t^i\), obtaining
\[ 72t^5 + 32t^4 + 64t^3 - 32t^2 + 48t^1 - 63t^0. \]

Carry: \[ 8t^6 - 4t^5 - 2t^4 + \]
\[ t^3 + 2t^2 + 2t^1 - 3t^0. \]

To minimize poly degree,
mix reduction and carrying,
carrying the top sooner.
\[ \text{e.g. Start from square} \]
\[ 25t^8 + 14t^7 + 48t^6 \]
\[ + 82t^3 + 43t^2 + 90t^1 + 81t^0. \]

Reduce \( t^{10} \rightarrow t^4 \):
\[ 56t^6 - 5t^5 + 2t^4 - \]
\[ 90t^1 + 81t^0. \]

Finish reduction: 
\[ 64t^3 - 32t^2 + 48t^1 - 63t^0. \]
Hmmm, is adjustment so easy?
Conditional branches are slow
and leak secrets through timing.

Can eliminate the branches,
but adjustment isn’t free.

Speedup: Skip the adjustment
for intermediate results.

“Lazy reduction.”
Adjust only for output.

\( b - 3a \) is small enough
to continue computations.

Can delay carries until after
multiplication by 3.

e.g. To square 314159
in \( \mathbb{Z}/1000003 \): Square poly

\[ 3t^5 + 1t^4 + 4t^3 + 1t^2 + 5t^1 + 9t^0, \]

obtaining \( 9t^{10} + 6t^9 + 25t^8 +
14t^7 + 48t^6 + 72t^5 + 59t^4 + \]
\( 82t^3 + 43t^2 + 90t^1 + 81t^0. \)

Reduce: replace \((c_i)t^{6+i}\) by
\((-3c_i)t^{i}\), obtaining
\( 72t^5 + 32t^4 + 64t^3 - 32t^2 + 48t^1 - 63t^0. \)

Carry: \( 8t^6 - 4t^5 - 2t^4 +
1t^3 + 2t^2 + 2t^1 - 3t^0. \)

To minimize poly degree,
mix reduction and carrying,
carrying the top sooner.

e.g. Start from square \( 9t^{10} +
25t^8 + 14t^7 + 48t^6 + 72t^5 +
82t^3 + 43t^2 + 90t^1 + 81t^0. \)

Reduce \( t^{10} \rightarrow t^4 \) and carry
\( t^5 \rightarrow t^6: 6t^9 + 25t^8 + 14t^7 +
56t^6 - 5t^5 + 2t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0. \)

Finish reduction: \(-5t^5 + 2t^4 +
64t^3 - 32t^2 + 48t^1 - 87t^0. \)

\( t^0 \rightarrow t^1 \rightarrow t^2 \rightarrow t^3 \rightarrow t^4 \rightarrow
-4t^5 - 2t^4 + 1t^3 + 2t^2 - 1t^1 \).
Can delay carries until after multiplication by 3.

e.g. To square 314159
in \( \mathbb{Z}/1000003 \): Square poly
\( 3t^5 + 1t^4 + 4t^3 + 1t^2 + 5t^1 + 9t^0 \),
obtaining \( 9t^{10} + 6t^9 + 25t^8 + 14t^7 + 48t^6 + 72t^5 + 59t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0 \).

Reduce: replace \((c_i)t^{6+i}\) by \((-3c_i)t^i\), obtaining \( 72t^5 + 32t^4 + 64t^3 - 32t^2 + 48t^1 - 63t^0 \).

Carry: \( 8t^6 - 4t^5 - 2t^4 + 1t^3 + 2t^2 + 2t^1 - 3t^0 \).

To minimize poly degree, mix reduction and carrying, carrying the top sooner.

e.g. Start from square \( 9t^{10} + 6t^9 + 25t^8 + 14t^7 + 48t^6 + 72t^5 + 59t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0 \).

Reduce \( t^{10} \to t^4 \) and carry \( t^4 \to t^5 \to t^6: 6t^9 + 25t^8 + 14t^7 + 56t^6 - 5t^5 + 2t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0 \).

Finish reduction: \(-5t^5 + 2t^4 + 64t^3 - 32t^2 + 48t^1 - 87t^0 \). Carry \( t^0 \to t^1 \to t^2 \to t^3 \to t^4 \to t^5: -4t^5 - 2t^4 + 1t^3 + 2t^2 - 1t^1 + 3t^0 \).
Can delay carries until after multiplication by 3.

e.g. To square 314159 in \( \mathbb{Z} = 1000003 \): Square poly

\( 3t^5 + 1t^4 + 4t^3 + 1t^2 + 5t^1 + 9t^0, \)

\( 9t^{10} + 6t^9 + 25t^8 + 48t^6 + 72t^5 + 59t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0. \)

Reduce \( t^{10} \rightarrow t^4 \) and carry \( t^4 \rightarrow t^5 \rightarrow t^6: 6t^9 + 25t^8 + 14t^7 + 56t^6 - 5t^5 + 2t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0. \)

Finish reduction: \(-5t^5 + 2t^4 + 64t^3 - 32t^2 + 48t^1 - 87t^0. \)

Carry \( t^0 \rightarrow t^1 \rightarrow t^2 \rightarrow t^3 \rightarrow t^4 \rightarrow t^5: -4t^5 - 2t^4 + 1t^3 + 2t^2 - 1t^1 + 3t^0. \)

To minimize poly degree, mix reduction and carrying, carrying the top sooner.

e.g. Start from square 9 \( t^{10} + 6t^9 + 25t^8 + 48t^6 + 72t^5 + 59t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0. \)

Reduce \( t^{10} \rightarrow t^4 \) and carry \( t^4 \rightarrow t^5 \rightarrow t^6: 6t^9 + 25t^8 + 14t^7 + 56t^6 - 5t^5 + 2t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0. \)

Finish reduction: \(-5t^5 + 2t^4 + 64t^3 - 32t^2 + 48t^1 - 87t^0. \)

Carry \( t^0 \rightarrow t^1 \rightarrow t^2 \rightarrow t^3 \rightarrow t^4 \rightarrow t^5: -4t^5 - 2t^4 + 1t^3 + 2t^2 - 1t^1 + 3t^0. \)

Speedup: non-integer radix \( p = 2^{61} - 1. \)

Five coeffs in radix 2
decide.

\( f_4t^4 + f_3t^3 + f_2t^2 + f_1t^1 + f_0t^0. \)

Most coeffs could be 2
twelve.

Square \( \cdots + f_4t^4 + f_3t^3 + f_2t^2 + f_1t^1 + f_0t^0. \)

Coeff of \( t^5 \) could be \( > 2^{25}. \)

Reduce: \( 2^{65} = 2^{4} \) in \( \mathbb{Z} = (2^{61} - 1); \)

\( \cdots + (2^{5}(f_4f_1 + f_3f_2) + f_2f_0)t^0. \)

Coeff could be \( > 2^{29}. \)

Very little room for additions, delayed carries, etc. on 32-bit platforms.
Can delay carries until after multiplication by 3.

To square 314159 in \( Z = 1000003 \): Square poly

\[
3t^5 + 1t^4 + 4t^3 + 1t^2 + 5t + 9t^0,
\]

obtaining

\[
9t^{10} + 6t^9 + 25t^8 + 14t^7 + 48t^6 + 72t^5 + 59t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0.
\]

Reduce \( t^{10} \rightarrow t^4 \) and carry \( t^4 \rightarrow t^5 \rightarrow t^6: 6t^9 + 25t^8 + 14t^7 + 56t^6 - 5t^5 + 2t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0. \)

Finish reduction: \(-5t^5 + 2t^4 + 64t^3 - 32t^2 + 48t^1 - 87t^0. \)

Carry \( t^0 \rightarrow t^1 \rightarrow t^2 \rightarrow t^3 \rightarrow t^4 \rightarrow t^5: -4t^5 - 2t^4 + 1t^3 + 2t^2 - 1t^1 + 3t^0. \)

To minimize poly degree, mix reduction and carrying, carrying the top sooner.

e.g. Start from square \( 9t^{10} + 6t^9 + 25t^8 + 14t^7 + 48t^6 + 72t^5 + 59t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0. \)

Reduce \( t^{10} \rightarrow t^4 \) and carry \( t^4 \rightarrow t^5 \rightarrow t^6: 6t^9 + 25t^8 + 14t^7 + 56t^6 - 5t^5 + 2t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0. \)

Finish reduction: \(-5t^5 + 2t^4 + 64t^3 - 32t^2 + 48t^1 - 87t^0. \)

Carry \( t^0 \rightarrow t^1 \rightarrow t^2 \rightarrow t^3 \rightarrow t^4 \rightarrow t^5: -4t^5 - 2t^4 + 1t^3 + 2t^2 - 1t^1 + 3t^0. \)

Speedup: non-integer radix \( p = 2^{61} - 1. \)

Five coeffs in radix \( 2^{13} \)?

\( f_4 t^4 + f_3 t^3 + f_2 t^2 \)

Most coeffs could be \( 2^{12} \).

Square \( \cdots + 2(f_4 f_1 + f_3 f_2) t^5 + \cdots + (2^5 (f_4 f_1 + f_3 f_2) + f_2) t^0. \)

Coeff of \( t^5 \) could be \( > 2^{25} \).

Reduce: \( 2^{65} = 2^4 \cdots + (2^5 (f_4 f_1 + f_3 f_2) + f_2) t^0. \)

Coeff could be \( > 2^{29} \).

Very little room for additions, delayed carries, etc. on 32-bit platforms.
To minimize poly degree, mix reduction and carrying, carrying the top sooner.

e.g. Start from square $9t^{10} + 6t^9 + 25t^8 + 14t^7 + 48t^6 + 72t^5 + 59t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0$.

Reduce $t^{10} \rightarrow t^4$ and carry $t^4 \rightarrow t^5 \rightarrow t^6$: $6t^9 + 25t^8 + 14t^7 + 56t^6 - 5t^5 + 2t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0$.

Finish reduction: $-5t^5 + 2t^4 + 64t^3 - 32t^2 + 48t^1 - 87t^0$. Carry $t^0 \rightarrow t^1 \rightarrow t^2 \rightarrow t^3 \rightarrow t^4 \rightarrow t^5$: $-4t^5 - 2t^4 + 1t^3 + 2t^2 - 1t^1 + 3t^0$.

Speedup: non-integer radix $p = 2^{61} - 1$.

Five coeffs in radix $2^{13}$?
$f_4t^4 + f_3t^3 + f_2t^2 + f_1t^1 + t^0$.

Most coeffs could be $2^{12}$.

Square $\cdots + 2(f_4f_1 + f_3f_2)t^5$.

Coeff of $t^5$ could be $> 2^{25}$.

Reduce: $2^{65} = 2^4$ in $\mathbb{Z}/(2^{61} - 1)$.

$\cdots + (2^5(f_4f_1 + f_3f_2) + f_0^2)t^0$.

Coeff could be $> 2^{29}$.

Very little room for additions, delayed carries, etc. on 32-bit platforms.
To minimize poly degree, mix reduction and carrying, carrying the top sooner.

e.g. Start from square $9t^{10} + 6t^9 + 25t^8 + 14t^7 + 48t^6 + 72t^5 + 59t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0$.

Reduce $t^{10} \rightarrow t^4$ and carry $t^4 \rightarrow t^5 \rightarrow t^6$: $6t^9 + 25t^8 + 14t^7 + 56t^6 - 5t^5 + 2t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0$.

Finish reduction: $-5t^5 + 2t^4 + 64t^3 - 32t^2 + 48t^1 - 87t^0$. Carry $t^0 \rightarrow t^1 \rightarrow t^2 \rightarrow t^3 \rightarrow t^4 \rightarrow t^5$: $-4t^5 - 2t^4 + 1t^3 + 2t^2 - 1t^1 + 3t^0$.

Speedup: non-integer radix

$p = 2^{61} - 1$.

Five coeffs in radix $2^{13}$?
$f_4 t^4 + f_3 t^3 + f_2 t^2 + f_1 t^1 + f_0 t^0$.

Most coeffs could be $2^{12}$.

Square $\cdots + 2(f_4 f_1 + f_3 f_2) t^5 + \cdots$.

Coeff of $t^5$ could be $> 2^{25}$.

Reduce: $2^{65} = 2^4$ in $\mathbb{Z}/(2^{61} - 1)$; $\cdots + (2^5(f_4 f_1 + f_3 f_2) + f_2^2) t^0$.

Coeff could be $> 2^{29}$.

Very little room for additions, delayed carries, etc. on 32-bit platforms.
To minimize poly degree, mix reduction and carrying, carrying the top sooner.

Start from square $9t^{10} + 6t^9 + 4t^7 + 48t^6 + 72t^5 + 59t^4 + 43t^2 + 90t^1 + 81t^0$.

$t^{10} \rightarrow t^4$ and carry $t^4 \rightarrow 6t^9 + 25t^8 + 14t^7 + 6t^5 + 2t^4 + 82t^3 + 43t^2 + 81t^0$.

Reduction: $-5t^5 + 2t^4 + 32t^2 + 48t^1 - 87t^0$. Carry $t^4 \rightarrow t^2 \rightarrow t^3 \rightarrow t^4 \rightarrow t^5$: $2t^4 + 1t^3 + 2t^2 - 1t^1 + 3t^0$.

---

**Speedup: non-integer radix**

$p = 2^{61} - 1$.

Five coeffs in radix $2^{13}$?

$f_4 t^4 + f_3 t^3 + f_2 t^2 + f_1 t^1 + f_0 t^0$.

Most coeffs could be $2^{12}$.

Square $\cdots + 2(f_4 f_1 + f_3 f_2) t^5 + \cdots$.

Coeff of $t^5$ could be $> 2^{25}$.

Reduce: $2^{65} = 2^4$ in $\mathbb{Z}/(2^{61} - 1)$;

$\cdots + (2^5 (f_4 f_1 + f_3 f_2) + f_2^2) t^0$.

Coeff could be $> 2^{29}$.

Very little room for additions, delayed carries, etc. on 32-bit platforms.

---

**Scaled:**

Evaluate at $t = 1$.

$f_4$ is multiple of $2^{52}$;

$f_3$ is multiple of $2^{39}$;

$f_2$ is multiple of $2^{26}$;

$f_1$ is multiple of $2^{13}$;

$f_0$ is multiple of $2^0$.

Saves a few bits in coeffs.
To minimize poly degree, mix reduction and carrying, carrying the top sooner.

\[ 9t^{10} + 6t^9 + 5 + 72t^5 + 59t^4 + 81t^1 + 81t^0. \]

Square and carry \( t^4 \) \( \rightarrow \)
\[ 5t^8 + 14t^7 + \cdots + 2(f_4f_1 + f_3f_2)t^5 + \cdots. \]

Coeff of \( t^5 \) could be > \( 2^{25} \).

Reduce: \( 2^{65} = 2^4 \) in \( \mathbb{Z}/(2^{61} - 1) \);
\[ \cdots + (2^5(f_4f_1 + f_3f_2) + f_2^2)t^0. \]

Coeff could be > \( 2^{29} \). Very little room for additions, delayed carries, etc. on 32-bit platforms.

**Speedup: non-integer radix**

\[ p = 2^{61} - 1. \]

Five coeffs in radix \( 2^{13} \)? \( f_4t^4 + f_3t^3 + f_2t^2 + f_1t^1 + f_0t^0. \)
Most coeffs could be \( 2^{12} \).

**Scaled:** Evaluate at \( t = 1. \)

\( f_4 \) is multiple of \( 2^{52} \);
\( f_3 \) is multiple of \( 2^{39} \);
\( f_2 \) is multiple of \( 2^{26} \);
\( f_1 \) is multiple of \( 2^{13} \);
\( f_0 \) is multiple of \( 2^0 \). Reduce:
\[ \cdots + (2^{-60}(f_4f_1 + f_3f_2 + f_2^2)f_0)t^0. \]

Better: Non-integer radix \( 2^{12} \):
\( f_4 \) is multiple of \( 2^{49} \);
\( f_3 \) is multiple of \( 2^{37} \);
\( f_2 \) is multiple of \( 2^{25} \);
\( f_1 \) is multiple of \( 2^{13} \);
\( f_0 \) is multiple of \( 2^0 \).

Saves a few bits in coeffs.
To minimize poly degree, mix reduction and carrying, carrying the top sooner.

\[9t^{10} + 6t^9 + 59t^4 + 14t^7 + 48t^6 + 72t^5 + 59t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0.\]

Reduce \(t^{10} \rightarrow t^4\) and carry \(t^4 \rightarrow t^5\):

\[6t^9 + 25t^8 + 14t^7 + 56t^6 - 5t^5 + 2t^4 + 82t^3 + 43t^2 + 90t^1 + 81t^0.\]

Finish reduction:

\[-5t^5 + 2t^4 + 64t^3 - 32t^2 + 48t^1 - 87t^0.\]

Carry \(t^0 \rightarrow t^1 \rightarrow t^2 \rightarrow t^3 \rightarrow t^4 \rightarrow t^5\):

\[-4t^5 - 2t^4 + t^3 + 2t^2 - t^1 + 3t^0.\]

**Speedup: non-integer radix**

\(p = 2^{61} - 1.\)

Five coeffs in radix \(2^{13}\)?

\(f_4 t^4 + f_3 t^3 + f_2 t^2 + f_1 t^1 + f_0 t^0.\)

Most coeffs could be \(2^{12}\).

Square \(\cdots + 2(f_4 f_1 + f_3 f_2) t^5 + \cdots.\)

Coeff of \(t^5\) could be \(> 2^{25}\).

Reduce: \(2^{65} = 2^4\) in \(\mathbb{Z}/(2^{61} - 1);\)

\(\cdots + (2^5(f_4 f_1 + f_3 f_2) + f_0^2) t^0.\)

Coeff could be \(> 2^{29}\).

Very little room for additions, delayed carries, etc. on 32-bit platforms.

**Scaled: Evaluate at \(t = 1.\)**

\(f_4\) is multiple of \(2^{52};\)

\(f_3\) is multiple of \(2^{39};\)

\(f_2\) is multiple of \(2^{26};\)

\(f_1\) is multiple of \(2^{13};\)

\(f_0\) is multiple of \(2^0.\) Reduce:

\(\cdots + (2^{-60} (f_4 f_1 + f_3 f_2) + f_0^2) t^0.\)

Better: Non-integer radix \(2^{12}\):

\(f_4\) is multiple of \(2^{49};\)

\(f_3\) is multiple of \(2^{37};\)

\(f_2\) is multiple of \(2^{25};\)

\(f_1\) is multiple of \(2^{13};\)

\(f_0\) is multiple of \(2^0.\) Saves a few bits in coeffs.
Speedup: non-integer radix $p = 2^{61} - 1$.

Five coeffs in radix $2^{13}$?
$f_4 t^4 + f_3 t^3 + f_2 t^2 + f_1 t + f_0 t^0$.

Most coeffs could be $2^{12}$.

Square $\cdots + 2(f_4 f_1 + f_3 f_2) t^5 + \cdots$.

Coeff of $t^5$ could be $> 2^{25}$.

Reduce: $2^{65} = 2^4$ in $\mathbb{Z}/(2^{61} - 1)$;
$\cdots + (2^5(f_4 f_1 + f_3 f_2) + f_0^2) t^0$.

Coeff could be $> 2^{29}$.

Very little room for additions, delayed carries, etc.
on 32-bit platforms.

Scaled: Evaluate at $t = 1$.

$f_4$ is multiple of $2^{52}$;
$f_3$ is multiple of $2^{39}$;
$f_2$ is multiple of $2^{26}$;
$f_1$ is multiple of $2^{13}$;
$f_0$ is multiple of $2^0$. Reduce:
$\cdots + (2^{-60}(f_4 f_1 + f_3 f_2) + f_0^2) t^0$.

Better: Non-integer radix $2^{12.2}$.

$f_4$ is multiple of $2^{49}$;
$f_3$ is multiple of $2^{37}$;
$f_2$ is multiple of $2^{25}$;
$f_1$ is multiple of $2^{13}$;
$f_0$ is multiple of $2^0$.

Saves a few bits in coeffs.
non-integer radix

\[ p = 2^{61} - 1. \]

Five coeffs in radix 2^{13}?

\[ f_4 t^4 + f_3 t^3 + f_2 t^2 + f_1 t^1 + f_0 t^0. \]

Coeffs could be 2^{12}.

\[ \cdots + 2(f_4 f_1 + f_3 f_2) t^5 + \cdots. \]

\[ t^5 \text{ could be } > 2^{25}. \]

\[ 2^{65} = 2^4 \text{ in } \mathbb{Z}/(2^{61} - 1); \]

\[ 5(f_4 f_1 + f_3 f_2) + f_0^2) t^0. \]

Could be > 2^{29}.

Very little room for additions, delayed carries, etc. on 32-bit platforms.

Scaled: Evaluate at \( t = 1. \)

\[ f_4 \text{ is multiple of } 2^{52}; \]

\[ f_3 \text{ is multiple of } 2^{39}; \]

\[ f_2 \text{ is multiple of } 2^{26}; \]

\[ f_1 \text{ is multiple of } 2^{13}; \]

\[ f_0 \text{ is multiple of } 2^0. \]

Reduce:

\[ \cdots + (2^{-60}(f_4 f_1 + f_3 f_2) + f_0^2) t^0. \]

Better: Non-integer radix 2^{12.2}.

\[ f_4 \text{ is multiple of } 2^{49}; \]

\[ f_3 \text{ is multiple of } 2^{37}; \]

\[ f_2 \text{ is multiple of } 2^{25}; \]

\[ f_1 \text{ is multiple of } 2^{13}; \]

\[ f_0 \text{ is multiple of } 2^0. \]

Saves a few bits in coeffs.

More bad choices from NIST

NIST P-256 prime:

\[ 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1. \]

i.e. \( t^8 - t^7 + t^6 + t^3 - 1 \) evaluated at \( t = 2^{32} \).
Non-integer radix

\[ p = 2^{61} - 1. \]

Five coeffs in radix 2

\[ f_4 t^4 + f_3 t^3 + f_2 t^2 + f_1 t^1 + f_0 t^0. \]

Most coeffs could be 2^{12}.

Square

\[ \cdots + 2(f_4 f_1 + f_3 f_2) t^5 + \cdots \]

Coeff of \( t^5 \) could be > 2^{25}.

Reduce:

\[ \cdots + (2^{-60}(f_4 f_1 + f_3 f_2) + f_0^2) t^0. \]

Better: Non-integer radix 2^{12.2}.

\[ f_4 \text{ is multiple of } 2^{49}; \]

\[ f_3 \text{ is multiple of } 2^{37}; \]

\[ f_2 \text{ is multiple of } 2^{25}; \]

\[ f_1 \text{ is multiple of } 2^{13}; \]

\[ f_0 \text{ is multiple of } 2^0. \]

Saves a few bits in coeffs.

Scaled: Evaluate at \( t = 1. \)

\[ f_4 \text{ is multiple of } 2^{52}; \]

\[ f_3 \text{ is multiple of } 2^{39}; \]

\[ f_2 \text{ is multiple of } 2^{26}; \]

\[ f_1 \text{ is multiple of } 2^{13}; \]

\[ f_0 \text{ is multiple of } 2^0. \]

Saves a few bits in coeffs.

More bad choices from NIST

NIST P-256 prime:

\[ 2^{256} - 2^{224} + 2^{192} - 1. \]

i.e. \( t^8 - t^7 + t^6 + t^3 - 1 \) evaluated at \( t = 2^{32}. \)
Scaled: Evaluate at $t = 1$.

- $f_4$ is multiple of $2^{52}$;
- $f_3$ is multiple of $2^{39}$;
- $f_2$ is multiple of $2^{26}$;
- $f_1$ is multiple of $2^{13}$;
- $f_0$ is multiple of $2^0$. Reduce:

$$\cdots + (2^{-60}(f_4 f_1 + f_3 f_2) + f_0^2)t^0.$$ 

Better: Non-integer radix $2^{12.2}$.

- $f_4$ is multiple of $2^{49}$;
- $f_3$ is multiple of $2^{37}$;
- $f_2$ is multiple of $2^{25}$;
- $f_1$ is multiple of $2^{13}$;
- $f_0$ is multiple of $2^0$.

Saves a few bits in coeffs.

More bad choices from NIST:

NIST P-256 prime:

$$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1.$$ 

i.e. $t^8 - t^7 + t^6 + t^3 - 1$ evaluated at $t = 2^{32}$. 

\[ \begin{align*} 
2^{256} &- 2^{224} + 2^{192} + 2^{96} - 1, \\
&\text{i.e. } t^8 - t^7 + t^6 + t^3 - 1 \\
&\text{evaluated at } t = 2^{32}. 
\end{align*} \]
Scaled: Evaluate at $t = 1$.

$f_4$ is multiple of $2^{52}$;
$f_3$ is multiple of $2^{39}$;
$f_2$ is multiple of $2^{26}$;
$f_1$ is multiple of $2^{13}$;
$f_0$ is multiple of $2^0$. Reduce:

$$\cdots + (2^{-60} (f_4 f_1 + f_3 f_2) + f_0^2) t^0.$$ 

Better: Non-integer radix $2^{12.2}$.

$f_4$ is multiple of $2^{49}$;
$f_3$ is multiple of $2^{37}$;
$f_2$ is multiple of $2^{25}$;
$f_1$ is multiple of $2^{13}$;
$f_0$ is multiple of $2^0$.

Saves a few bits in coeffs.

More bad choices from NIST

NIST P-256 prime:

$$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1.$$ 

i.e. $t^8 - t^7 + t^6 + t^3 - 1$ evaluated at $t = 2^{32}$. 

Scaled: Evaluate at \( t = 1 \).
\( f_4 \) is multiple of \( 2^{52} \);
\( f_3 \) is multiple of \( 2^{39} \);
\( f_2 \) is multiple of \( 2^{26} \);
\( f_1 \) is multiple of \( 2^{13} \);
\( f_0 \) is multiple of \( 2^0 \). Reduce:
\[ \cdots + (2^{-60}(f_4 f_1 + f_3 f_2) + f_0^2)t^0. \]

Better: Non-integer radix \( 2^{12.2} \).
\( f_4 \) is multiple of \( 2^{49} \);
\( f_3 \) is multiple of \( 2^{37} \);
\( f_2 \) is multiple of \( 2^{25} \);
\( f_1 \) is multiple of \( 2^{13} \);
\( f_0 \) is multiple of \( 2^0 \).
Saves a few bits in coeffs.

More bad choices from NIST

NIST P-256 prime:
\[ 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1. \]
i.e. \( t^8 - t^7 + t^6 + t^3 - 1 \)
evaluated at \( t = 2^{32} \).

Reduction: replace \( c_i t^{8+i} \) with \( c_i t^{7+i} - c_i t^{6+i} - c_i t^{3+i} + c_i t^i \).
Minor problem: often slower than small const mult and one add.
Scaled: Evaluate at $t = 1$.

$f_4$ is multiple of $2^{52}$;
$f_3$ is multiple of $2^{39}$;
$f_2$ is multiple of $2^{26}$;
$f_1$ is multiple of $2^{13}$;
$f_0$ is multiple of $2^0$. Reduce:

\[ \cdots + (2^{-60}(f_4 f_1 + f_3 f_2) + f_0^2) t^0. \]

Better: Non-integer radix $2^{12.2}$.

$f_4$ is multiple of $2^{49}$;
$f_3$ is multiple of $2^{37}$;
$f_2$ is multiple of $2^{25}$;
$f_1$ is multiple of $2^{13}$;
$f_0$ is multiple of $2^0$.

Saves a few bits in coeffs.

More bad choices from NIST

NIST P-256 prime:

$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.
i.e. $t^8 - t^7 + t^6 + t^3 - 1$
evaluated at $t = 2^{32}$.

Reduction: replace $c_i t^{8+i}$ with $c_i t^{7+i} - c_i t^{6+i} - c_i t^{3+i} + c_i t^i$.

Minor problem: often slower than small const mult and one add.

Major problem: With radix $2^{32}$, products are almost $2^{64}$.
Sums are slightly above $2^{64}$: bad for every common CPU.
Need very frequent carries.