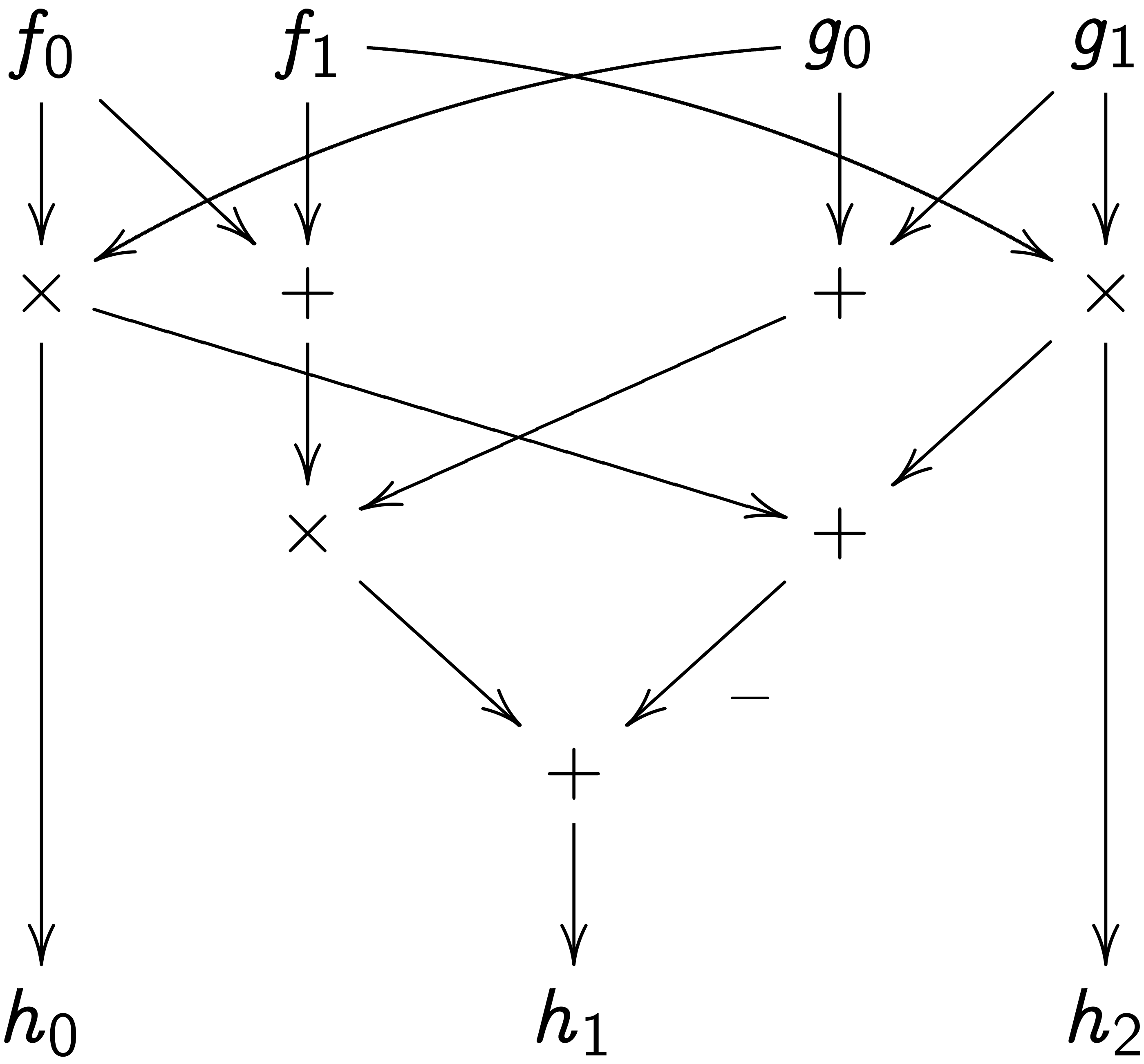
The tangent FFT

D. J. Bernstein
University of Illinois at Chicago

See online version of paper, particularly for bibliography:
`http://cr.yp.to/papers.html#tangentfft`

Algebraic algorithms



$\times$ multiplies its two inputs.

$+$ adds its two inputs.

$+^-$ subtracts its two inputs.

This "**R**-algebraic algorithm" computes product $h_0 + h_1 x + h_2 x^2$ of $f_0 + f_1 x, g_0 + g_1 x \in \mathbf{R}[x]$.

More precisely: It computes the coeffs of the product (on standard basis $1, x, x^2$) given the coeffs of the factors (on standard bases $1, x$ and $1, x$).

3 mults, 4 adds.
Compare to obvious algorithm:
4 mults, 1 add.

(1963 Karatsuba)

Algebraic complexity

Are 3 mults, 4 adds
better than 4 mults, 1 add?

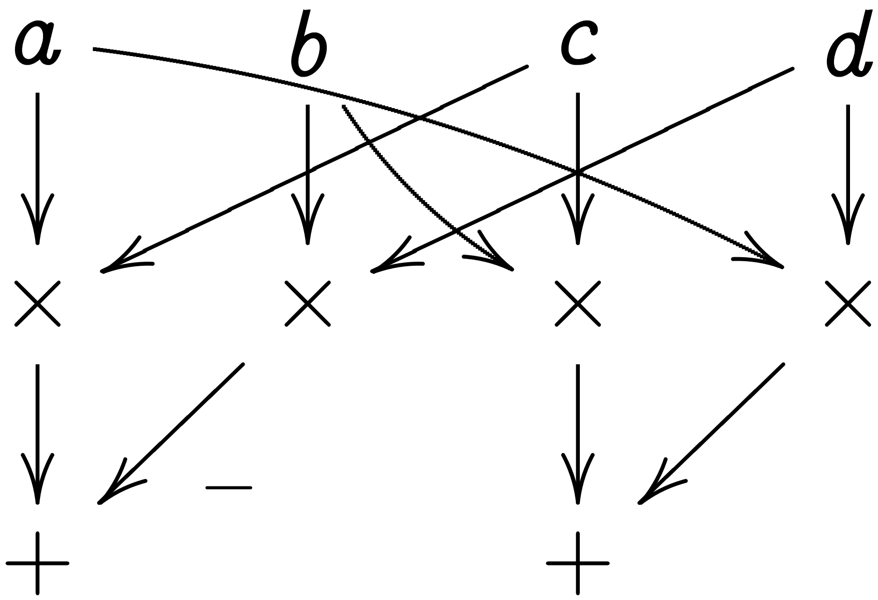
In this talk: No!

Cost measure for this talk:
"total **R**-algebraic complexity."
$+$ ("add"): cost 1.
$+^-$ (also "add"): cost 1.
$\times$ ("mult"): cost 1.
Constant in **R**: cost 0.

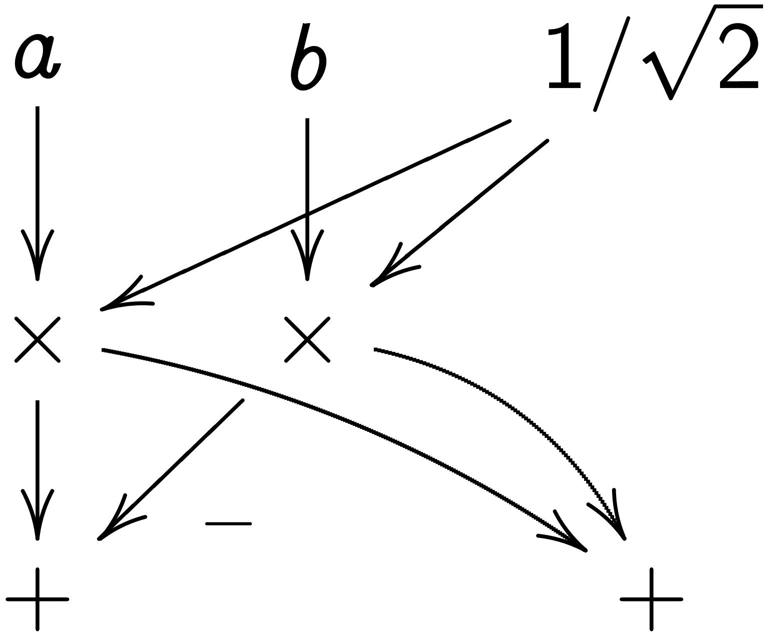
3 mults, 4 adds: cost 7.
4 mults, 1 add: cost 5.

Cost 6 to multiply in **C**

(on standard basis $1, i$):



Cost 4 to multiply by $\sqrt{i}$:



Can use (e.g.) Pentium M's
80-bit floating-point instructions
to approximate operations in **R**.

Each cycle, Pentium M follows
≤ 1 floating-point instruction.
So #Pentium M cycles
$\geq$ total **R**-algebraic complexity.

Usually can achieve #cycles
$\approx$ total **R**-algebraic complexity.
Analysis of "usually" and "$\approx$"
is beyond this talk.

Many other cost measures.

Some measures emphasize adds.
e.g. 64-bit fp on one core of Core 2 Duo: #cycles $\approx \max\{\#\mathbf{R}\text{-adds}, \#\mathbf{R}\text{-mults}\}/2$.
Typically more adds than mults.

Some measures emphasize mults.
e.g. Dedicated hardware for floating-point arithmetic: mults more expensive than adds.

But "cost" in this talk means $\#\mathbf{R}\text{-adds} + \#\mathbf{R}\text{-mults}$.

Fast Fourier transforms

Define $\zeta_n \in \mathbf{C}$ as $\exp(2\pi i/n)$.
Define $T_n : \mathbf{C}[x]/(x^n - 1) \hookrightarrow\!\!\!\!\!\!\!\twoheadrightarrow \mathbf{C}^n$
as $f \mapsto f(1), f(\zeta_n), \ldots, f(\zeta_n^{n-1})$.

Can very quickly compute T_n.

First publication of fast algorithm: 1866 Gauss.

Easy to see that Gauss's FFT uses $O(n \lg n)$ arithmetic operations if $n \in \{1, 2, 4, 8, \ldots\}$.

Several subsequent reinventions, ending with 1965 Cooley/Tukey.

Inverse map is also very fast.

Multiplication in $\mathbf{C}^n$ is very fast.

1966 Sande, 1966 Stockham:
Can very quickly multiply
in $\mathbf{C}[x]/(x^n - 1)$ or $\mathbf{C}[x]$ or $\mathbf{R}[x]$
by mapping $\mathbf{C}[x]/(x^n - 1)$ to $\mathbf{C}^n$.
"Fast convolution."

Given $f, g \in \mathbf{C}[x]/(x^n - 1)$:
compute fg as $T_n^{-1}(T_n(f)T_n(g))$.

Given $f, g \in \mathbf{C}[x]$, $\deg fg < n$:
compute fg from
its image in $\mathbf{C}[x]/(x^n - 1)$.

Cost $O(n \lg n)$.

A closer look at costs

More precise analysis of Gauss FFT (and Cooley-Tukey FFT):

$\mathbf{C}[x]/(x^n - 1) \hookrightarrow\!\!\!\!\rightarrow \mathbf{C}^n$ using $n \lg n$ $\mathbf{C}$-adds (costing 2 each), $(n \lg n)/2$ $\mathbf{C}$-mults (6 each), if $n \in \{1, 2, 4, 8, \ldots\}$.

Total cost $5n \lg n$.
After peephole optimizations: cost $5n \lg n - 10n + 16$ if $n \in \{4, 8, 16, 32, \ldots\}$.

Either way, $5n \lg n + O(n)$.
This talk focuses on the 5.

What about cost of convolution?

$5n \lg n + O(n)$ to compute $T_n(f)$,
$5n \lg n + O(n)$ to compute $T_n(g)$,
$O(n)$ to multiply in $\mathbf{C}^n$,
similar $5n \lg n + O(n)$ for T_n^{-1}.

Total cost $15n \lg n + O(n)$
to compute $fg \in \mathbf{C}[x]/(x^n - 1)$
given $f, g \in \mathbf{C}[x]/(x^n - 1)$.

Total cost $(15/2)n \lg n + O(n)$
to compute $fg \in \mathbf{R}[x]/(x^n - 1)$
given $f, g \in \mathbf{R}[x]/(x^n - 1)$: map
$\mathbf{R}[x]/(x^n - 1) \hookrightarrow\!\!\!\!\rightarrow \mathbf{R}^2 \oplus \mathbf{C}^{n/2-1}$
(Gauss) to save half the time.

1968 R. Yavne: Can do better!

Cost $4n \lg n + O(n)$

to map $\mathbf{C}[x]/(x^n - 1) \hookrightarrow\!\!\!\!\rightarrow \mathbf{C}^n$,

if $n \in \{1, 2, 4, 8, 16, \ldots\}$.

1968 R. Yavne: Can do better!
Cost $4n \lg n + O(n)$
to map $\mathbf{C}[x]/(x^n - 1) \hookrightarrow\!\!\!\!\rightarrow \mathbf{C}^n$,
if $n \in \{1, 2, 4, 8, 16, \ldots\}$.

2004 James Van Buskirk:
Can do better!
Cost $(34/9)n \lg n + O(n)$.

Expositions of the new algorithm:
Frigo, Johnson,
in *IEEE Trans. Signal Processing*;
Lundy, Van Buskirk,
in *Computing*;
Bernstein, this *AAECC* paper.

Understanding the FFT

If $f \in \mathbf{C}[x]$ and
$f \bmod x^4 - 1 =$
$f_0 + f_1 x + f_2 x^2 + f_3 x^3$ then
$f \bmod x^2 - 1 =$
$(f_0 + f_2) + (f_1 + f_3)x$,
$f \bmod x^2 + 1 =$
$(f_0 - f_2) + (f_1 - f_3)x$.

Given $f \bmod x^4 - 1$,
cost 8 to compute
$f \bmod x^2 - 1$, $f \bmod x^2 + 1$.

"$\mathbf{C}[x]$-morphism $\mathbf{C}[x]/(x^4 - 1) \hookrightarrow\!\!\!\!\rightarrow$
$\mathbf{C}[x]/(x^2 - 1) \oplus \mathbf{C}[x]/(x^2 + 1)$."

If $f \in \mathbf{C}[x]$ and

$f \bmod x^{2n} - r^2 =$

$f_0 + f_1 x + \cdots + f_{2n-1} x^{2n-1}$ then

$f \bmod x^n - r =$

$$(f_0 + r f_n) + (f_1 + r f_{n+1})x \\ \quad + (f_2 + r f_{n+2})x^2 + \cdots,$$

$f \bmod x^n + r =$

$$(f_0 - r f_n) + (f_1 - r f_{n+1})x \\ \quad + (f_2 - r f_{n+2})x^2 + \cdots.$$

Given $f_0, f_1, \ldots, f_{2n-1} \in \mathbf{C}$,

cost $\le 10n$ to compute

$f_0 + r f_n, f_1 + r f_{n+1}, \ldots,$

$f_0 - r f_n, f_1 - r f_{n+1}, \ldots.$

Note: can compute in place.

The FFT: Do this recursively!

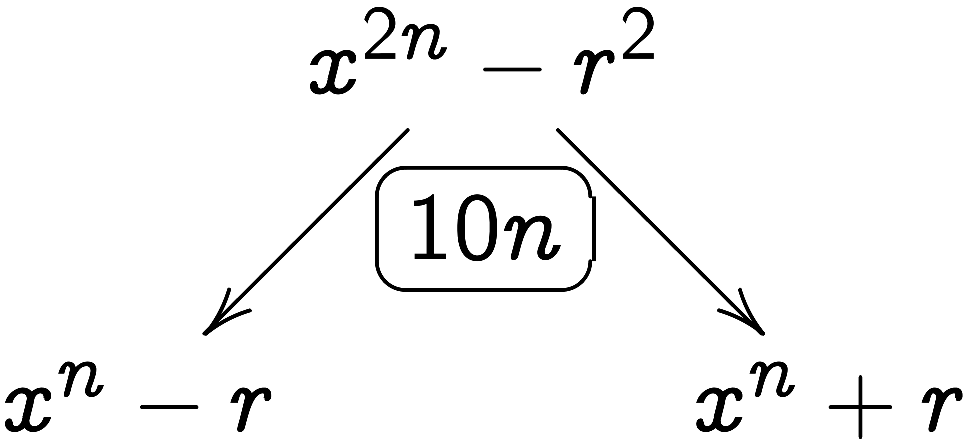
$$f \bmod x^4 - 1$$

$$f \bmod x^2 - 1 \qquad f \bmod x^2 + 1$$

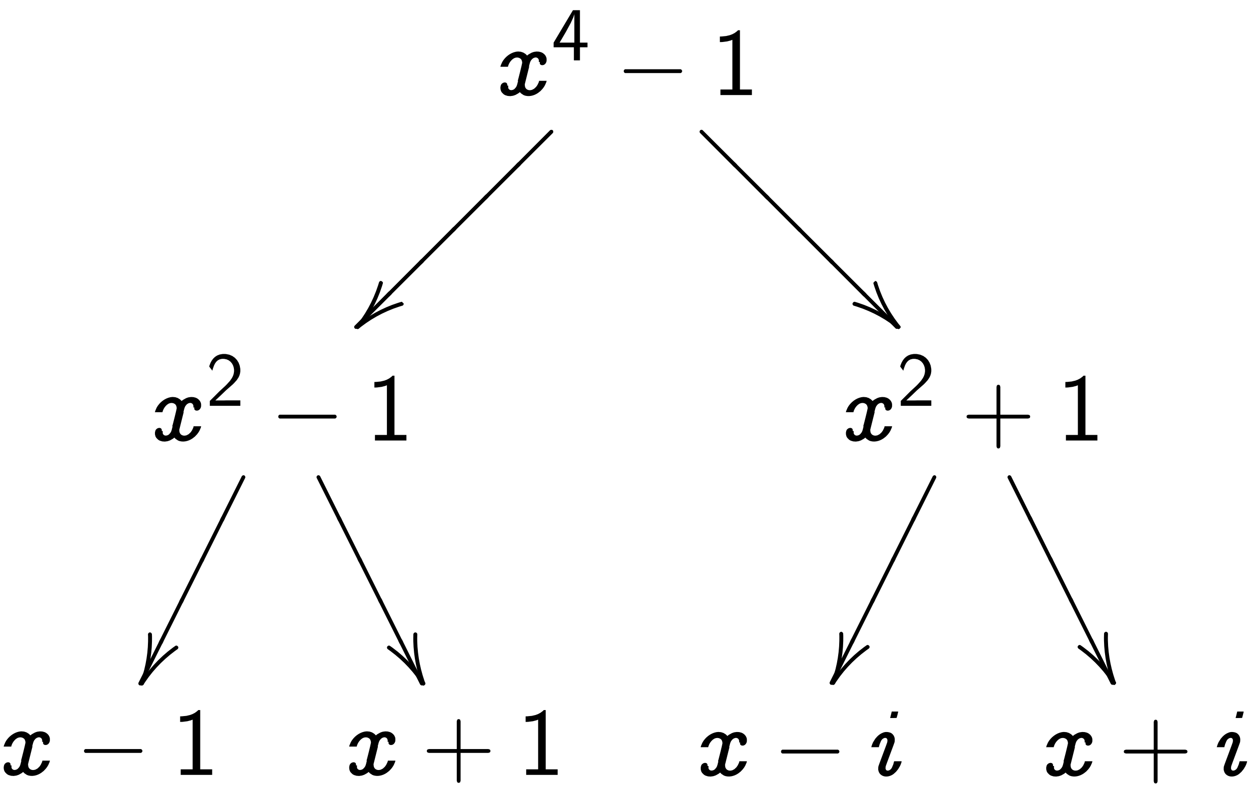
$$f \bmod x - 1 = f(1) \qquad f \bmod x + 1 = f(-1) \qquad f \bmod x - i = f(i) \qquad f \bmod x + i = f(-i)$$

(expository idea: 1972 Fiduccia)

Modulus tree for one step:



Modulus tree for full size-4 FFT:



Alternative: the twisted FFT

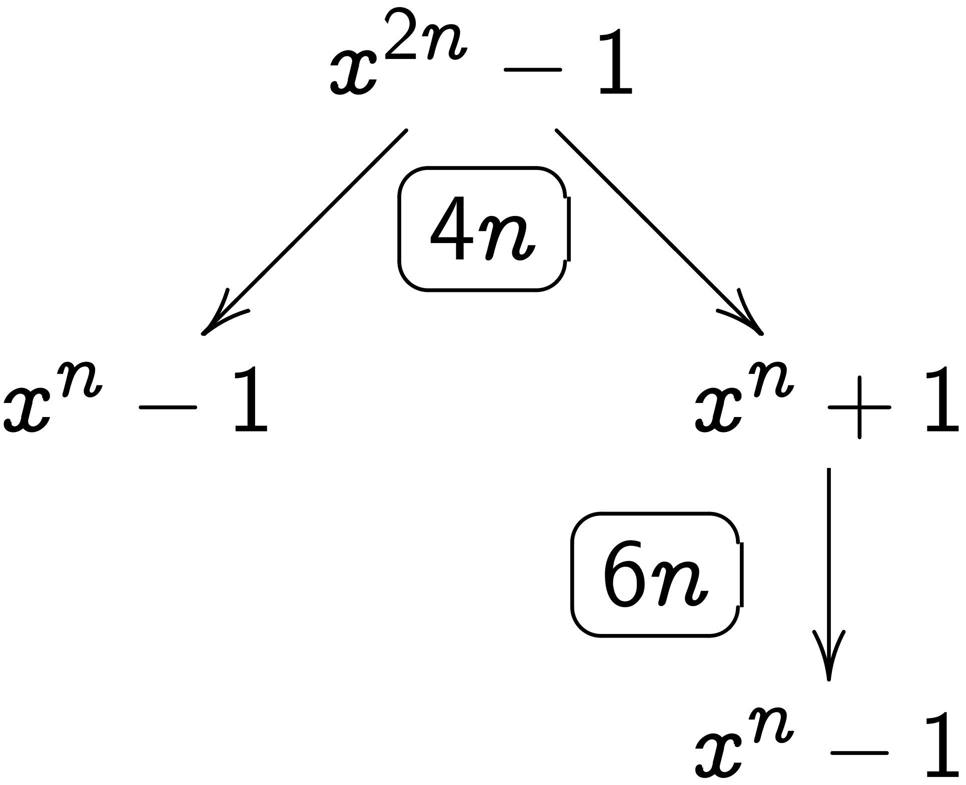
If $f \in \mathbf{C}[x]$ and
$f \bmod x^n + 1 =$
$g_0 + g_1 x + g_2 x^2 + \cdots$ then
$f(\zeta_{2n} x) \bmod x^n - 1 =$
$g_0 + \zeta_{2n} g_1 x + \zeta_{2n}^2 g_2 x^2 + \cdots$.

"$\mathbf{C}$-morphism $\mathbf{C}[x]/(x^n + 1) \hookrightarrow\!\!\!\!\rightarrow$
$\mathbf{C}[x]/(x^n - 1)$ by $x \mapsto \zeta_{2n} x$."

Modulus tree:

$$x^n + 1$$
$$\boxed{6n} \quad \Big\downarrow$$
$$x^n - 1$$

Merge with the original FFT trick:



"Twisted FFT" applies
this modulus tree recursively.

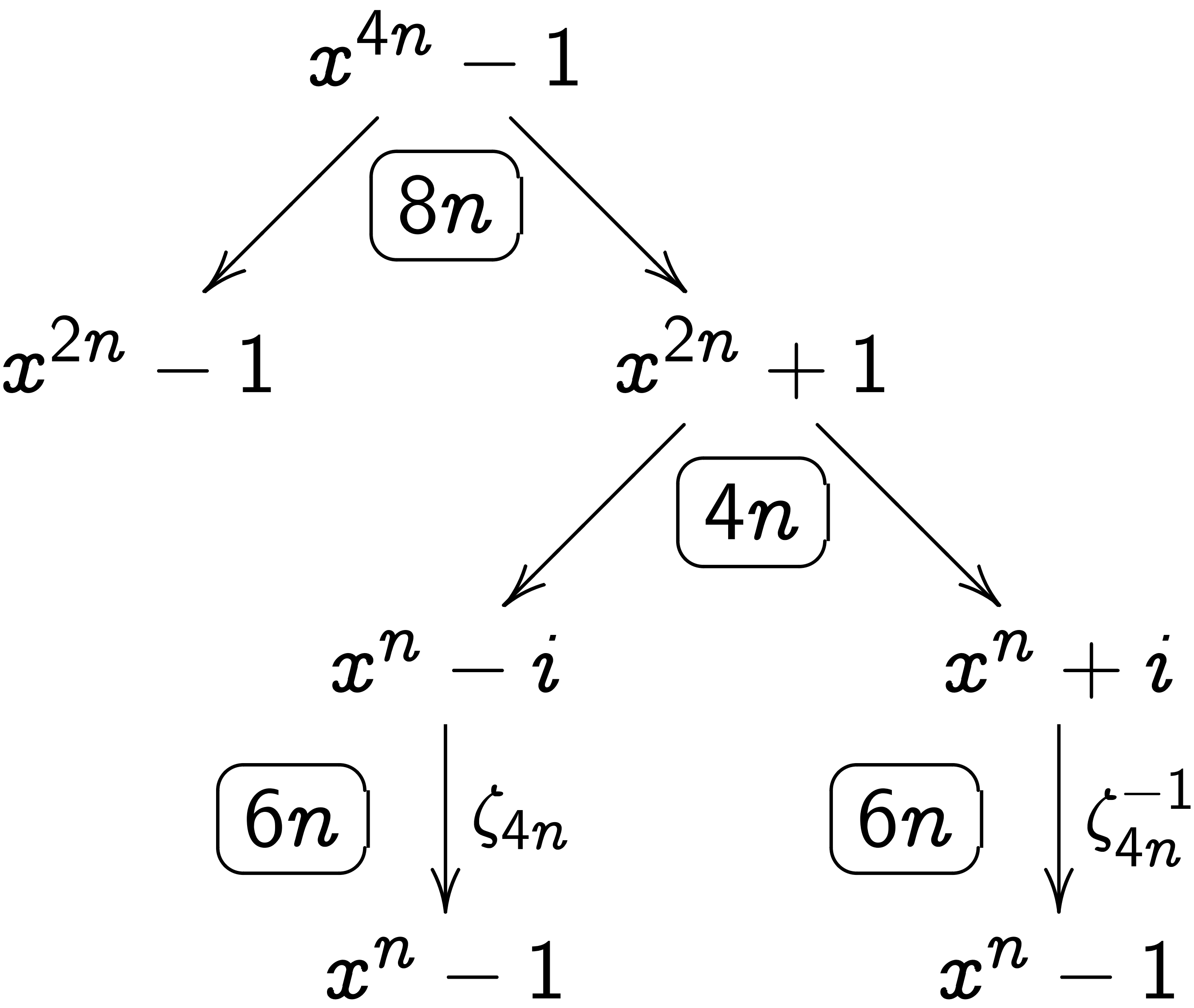
Cost $5n \lg n + O(n)$,
just like the original FFT.

The split-radix FFT

FFT and twisted FFT end up with
same number of mults by ζ_n,
same number of mults by $\zeta_{n/2}$,
same number of mults by $\zeta_{n/4}$,
etc.

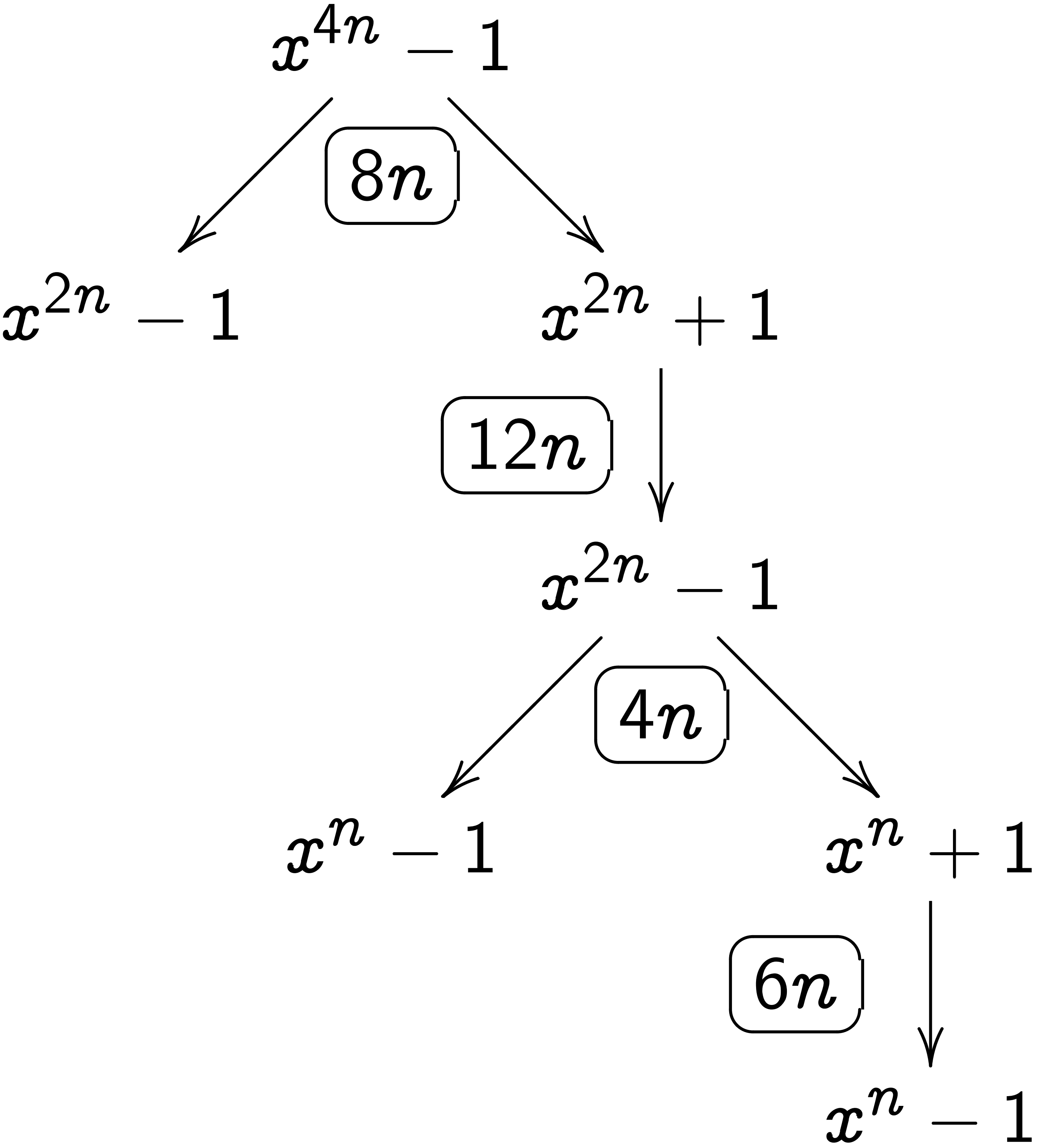
Is this necessary? No!
Split-radix FFT: more easy mults.
"Don't twist until you see
the whites of their i's."

(Can use same idea to speed up
Schönhage-Strassen algorithm
for integer multiplication.)



Split-radix FFT applies
this modulus tree recursively.
Cost $4n \lg n + O(n)$.

Compare to how twisted FFT splits $4n$ into $2n, n, n$:



The tangent FFT

Several ways to achieve
cost 6 for mult by $e^{i\theta}$.

One approach: Factor $e^{i\theta}$
as $(1 + i \tan\theta) \cos\theta$.
Cost 2 for mult by $\cos\theta$.
Cost 4 for mult by $1 + i \tan\theta$.

For stability and symmetry,
use $\max\{|\cos\theta|, |\sin\theta|\}$
instead of $\cos\theta$.

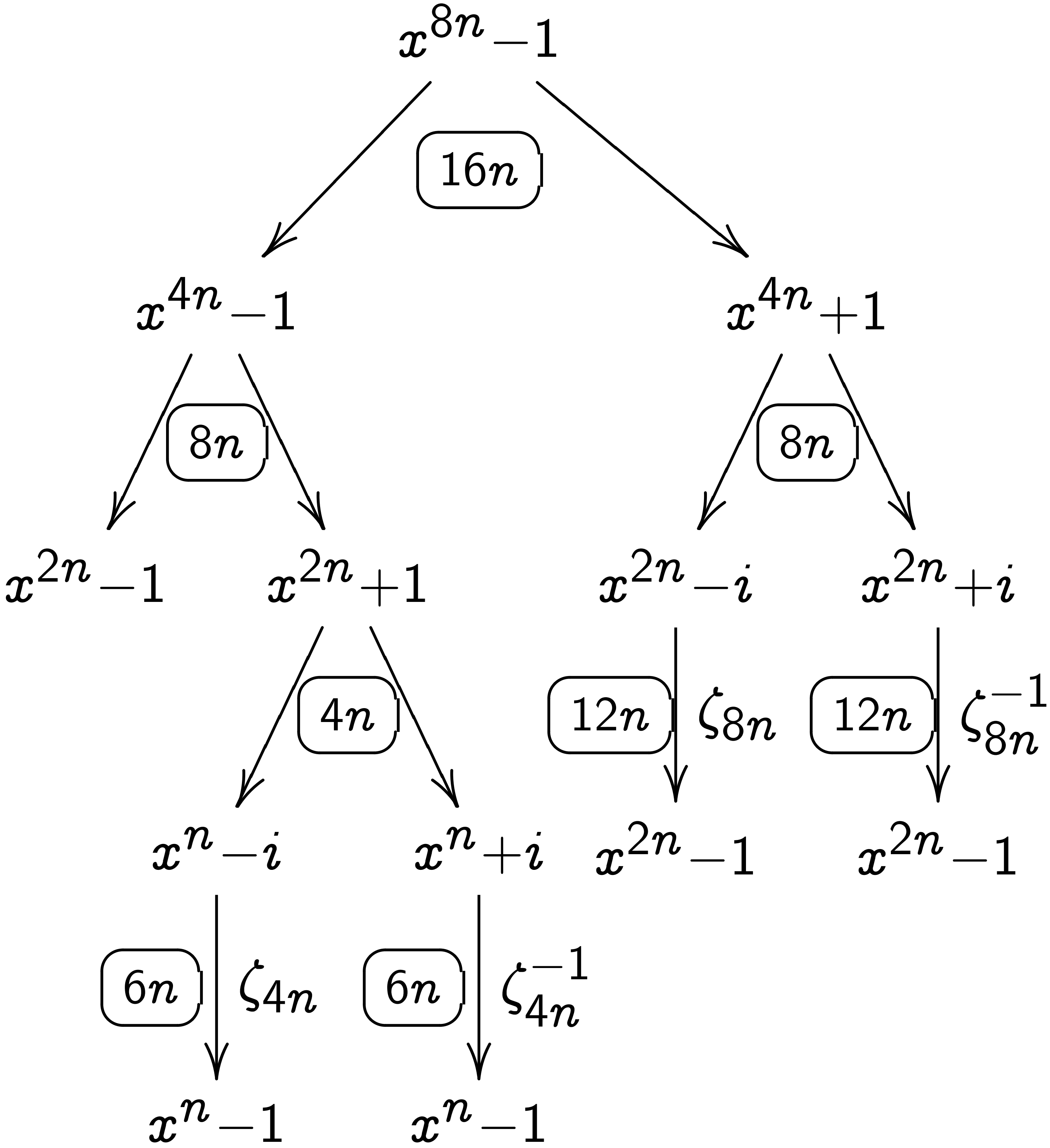
Surprise (Van Buskirk):
Can merge some cost-2 mults!

Rethink basis of $\mathbf{C}[x]/(x^n - 1)$.
Instead of $1, x, \ldots, x^{n-1}$ use
$1/s_{n,0}, x/s_{n,1}, \ldots, x^{n-1}/s_{n,n-1}$
where $s_{n,k} =$
$\max\{\left|\cos \frac{2\pi k}{n}\right|, \left|\sin \frac{2\pi k}{n}\right|\} \cdot$
$\max\{\left|\cos \frac{2\pi k}{n/4}\right|, \left|\sin \frac{2\pi k}{n/4}\right|\} \cdot$
$\max\{\left|\cos \frac{2\pi k}{n/16}\right|, \left|\sin \frac{2\pi k}{n/16}\right|\} \cdot$
$\cdots$.

Now $(g_0, g_1, \ldots, g_{n-1})$ represents
$g_0/s_{n,0} + \cdots + g_{n-1}x^{n-1}/s_{n,n-1}$.

Note that $s_{n,k} = s_{n,k+n/4}$.
Note that $\zeta_n^k(s_{n/4,k}/s_{n,k})$ is
$\pm(1 + i \tan \cdots)$ or $\pm(\cot \cdots + i)$.

Look at how split-radix splits $8n$ into $2n, 2n, 2n, n, n$:



New basis saves $12n$:
$4n$ in ζ_{8n} twist, $4n$ in ζ_{8n}^{-1} twist,
$2n$ in ζ_{4n} twist, $2n$ in ζ_{4n}^{-1} twist.

New basis costs $8n$:
$4n$ to change basis of $x^{2n} + 1$,
$4n$ to change basis
of top-left $x^{2n} - 1$.

Overall $68n$ instead of $72n$.
Recurse: $(34/9)n \lg n + O(n)$,
as in 2004 Van Buskirk.

Open: Can 34/9 be improved?