COS 226 Lecture 21: Multiplication

How fast can we multiply?

Small numbers: basic machine operation
Approximate solution OK: "floating point"
    MIPS vs. FLOPS
Exact solution required for numbers
    with millions of digits?

small ex:

    98734019238470981723409182730491827
        times
    66753875434605462654864811227186687
        equals
    6590878421402831216001745213641931592836606299109
            07873145346994326436

Basic question about difficulty of computations

ARITHMETIC COMPLEXITY
    also consider other operations
    add, multiply, exponentiate, divide, logarithm ...

Applications more numerous than we might imagine
    Arithmetic coding ($%#?!)
    Cryptography
    Signal processing

Multiplying complex numbers

Given two complex numbers

```
     x = a + b i
     y = c + d i
```

Product x*y

```
     real part:      a*c - b*d
     imaginary part: a*d + b*c
```

Four multiplications, two additions

Typical old-fashioned machine:

```
 1 time unit for add
 4 time units for multiply
 therefore, 18 time units for complex multiply
```

Faster way to compute product (!)

```
     t = (a+b) * (c+d) = a*c + a*d + b*c + b*d
```

Product x*y

```
     real part:      a*c - b*d
     imaginary part: t - a*c - b*d
```

THREE multiplications, five additions

```
     17 time units for complex multiply
```

Only works if multiply expensive relative to add

```
     software floating point
     old machines
```

Exponentiation

How many multiplications to compute x^N ?

If N is a power of 2, use successive squaring
    1, x, x^2, x^4, x^8, x^16, x^32, ...

If N is not a power of 2, do successive squaring
    then pick out numbers to multiply together by
    * divide-and-conquer
    * binary representation of N

Ex: x = 2, N = 21

1 2 4 8 16 32 64 128 256 512 1024 2048 4096
8192 16384 32768 65536

binary rep. of N: 2^21 = 2^16*2^4*2^1 = 65536*16*2 =
divide-and-conquer: 2^21 = 2^10*2^11 = 2^5*2*5*2*5*2^6 = ...

Unrealistic to consider if x and/or N large

Realistic version (which is of practical use)
    THM: Less than 2 lg N multiplications
    are required to compute x^N mod M

Linear in the number of bits to store N

DISCRETE LOGARITHM problem (difficult)
    given x^N mod M, find x

Multiplying polynomials

To multiply integers

may as well consider polynomial multiplication

finish up with Horner's method

grade school

```
.       1 2 3 4
.       5 6 7 8
.       -------
.       9 8 7 2
.     8 6 3 8
.   7 4 0 4
. 6 1 7 0
. -------------
. 7 0 0 6 6 5 2
```

grade school, no carries

```
                1   2   3   4
                5   6   7   8
               ---------------
               8*1 8*2 8*3 8*4
           7*1 7*2 7*3 7*4
       6*1 6*2 6*3 6*4
   5*1 5*2 5*3 5*4
   ---------------------------
```

polynomial

$$
\begin{array}{lrrrrrrr}
: & & & & 1x^3 & + 2x^2 & + 3x^1 & + 4x^0 \\
: & & & & 5x^3 & + 6x^2 & + 7x^1 & + 8x^0 \\
. & & & & \text{-----} & \text{-----} & \text{-----} & \text{-----} \\
: & & & & 8x^3 & +16x^2 & +24x^1 & +32x^0 \\
: & & & 7x^4 & +14x^3 & +21x^2 & +28x^1 & \\
: & & 6x^5 & +12x^4 & +18x^3 & +24x^2 & & \\
: & 5x^6 & +10x^5 & +15x^4 & +20x^3 & & & \\
. & \text{-----} & \text{-----} & \text{-----} & \text{-----} & \text{-----} & \text{-----} & \text{-----} \\
: & 5x^6 & +16x^5 & +34x^4 & +60x^3 & +61x^2 & +52x^1 & +32x^0
\end{array}
$$

Cost of multiplication

Polynomial multiplication

```
for (i = 0; i < 2*N-1; i++) r[i] = 0;
for (i = 0; i < N; i++)
   for (j = 0; j < N; j++)
      r[i+j] += p[i]*q[j];
```

N^2 coefficient multiplications

Integer multiplication

R–L:

```
.       32
.      52
.     61
.    60
.   34
.  16
.  5
.  7006652
```

L–R: use Horner's method to evaluate at x=10

```
.       5*10 =      50+16 = 66
.      66*10 =     660+34 = 6944
.     694*10 =    6940+60 = 7000
.    7000*10 =   70000+61 = 70061
.   70061*10 =  700610+52 = 700662
.  700662*10 = 7006620+32 = 7006652
```

Extra factor of log N might be required

Bottom line: quadratic algorithms
can multiply thousand–digit numbers
but not million–digit numbers

Divide-conquer polynomial multiplication

Example

```
: ( x^2( x+2) + (3x+4) ) * (  x^2(5x+6) + (7x+8) )

:  x^4( x+2)*(5x+6)
:           + x^2( x+2)*(7x+8)
:           + x^2(5x+6)*(3x+4)
.                      + (3x+4)*(7x+8)

: 5x^6 +16x^5 +12x^4
:             7x^4 +22x^3 +16x^2
:            15x^4 +38x^3 +24x^2
:                        21x^2 +52x  +32
. ----------------------------------------
: 5x^6 +16x^5 +34x^4 +60x^3 +61x^2 +52x  +32
```

In general, if p and q are degree 2N

```
:  p(x)*q(x) = (x^N ph(x)+pl(x))*(x^N qh(x)+ql(x))
:  p(x)*q(x) = x^2N ph(x)*qh(x)
:               + x^N(ph(x)*ql(x) + pl(x)*qh(x))
.               + pl(x)*ql(x)
```

Scalar multiplication count

     m(0) = 1

 m(2N) = 4*m(N)

Solution: m(N) = N^2, since (2N)^2 = 4N^2

No savings from divide and conquer?

Polynomial multiplication (continued)

Use same technique as for complex numbers

In general, if p and q are degree 2N

: $p(x)*q(x) = (x^N ph(x)+pl(x))*(x^N qh(x)+ql(x))$

. $t(x) = (ph(x) + pl(x)) * (qh(x) + ql(x))$

: $p(x)*q(x) = x^{2N} ph(x)*qh(x)$

: $+ x^N(t(x)-ph(x)*qh(x) + pl(x)*ql(x))$

. $+ pl(x)*ql(x)$

Ex:

: $(x^2(x+2) + (3x+4)) * (x^2(5x+6) + (7x+8))$

: $(x+2)*(5x+ 6) = 5x^2+ 16x + 12$

: $(3x+4)*(7x+ 8) = 21x^2+ 52x + 32$

: $t(x) = (6x+8)*(10x+12) = 60x^2+ 152x + 96$

: $x^4(5x^2+ 16x + 12)$

: $+ x^2(34x^2+84x+52)$

: $+ 21x^2+ 52x + 32$

. --

: $5x^6 +16x^5 +34x^4 +60x^3 +61x^2 + 52x + 32$

Scalar multiplication count

m(0) = 1

m(2N) = 3*m(N)

Solution: m(N) = N^lg3, since (2N)^lg3 = 3N^lg3

Bottom line:

can do tens of thousands of digits

but not millions of digits

Basic computations on polynomials

A polynomial of degree N-1

$$p0 + p1\,x + p2\,x^2 + p3\,x^3 + \ldots + p(N-1)\,x^{N-1}$$

is defined by N numbers (the coefficients)

p0 p1 p2 ... p(N-1)

The polynomial associates any N numbers

x0 x1 x2 ... x(N-1)

with another set of N numbers

y0 y1 y2 ... y(N-1)

simply by

p(x0) = y0
p(x1) = y1
p(x2) = y2
p(x3) = y3
...

EVALUATION

find the y's, given the p's and x's

ROOT EXTRACTION

find the x's, given the p's and y's

INTERPOLATION

find the p's, given the x's and y's

Fast polynomial multiplication

EVALUATE–MULTIPLY–INTERPOLATE

Ex: $(1+x)*(2+3x) = 2 + 5x + 3x^2$

Pick three distinct numbers: 0, 1, 2 (say)

EVALUATE

x	0	1	2
1 + x	1	2	3
2 + 3x	2	5	8

MULTIPLY

x	0	1	2
(1 + x)*(2 + 3x)	2	10	24

INTERPOLATE

$$2\,\frac{(x-1)(x-2)}{(0-1)(0-2)} + 10\,\frac{(x-0)(x-2)}{(1-0)(1-2)} + 24\,\frac{(x-0)(x-1)}{(2-0)(2-1)}$$

$$x^2 - 3x + 2 \; -10\,(x^2 - 2x) + 12\,(x^2 - x)$$

$$3x^2 + 5x + 2$$

Bad news

* evaluate, interpolate, seem to take N^2 steps

Good news

* method works for ANY set of distinct points

Complex roots of unity

Ex:

$(1-i)(1-i) = 1 - 2i - 1 = -2i$

$(1-i)^4 = 4$

For each N, there are N different complex numbers that evaluate to N when raised to the Nth power

: $w_8^0 \quad w_8^1 \quad w_8^2 \quad w_8^3 \quad w_8^4 \quad w_8^5 \quad w_8^6 \quad w_8^7$

. $1 \quad w \quad \quad \quad -1 \quad w^5$

w: PRINCIPAL Nth root of unity

: $W_N^k = e^{(2 \text{ pi } i \text{ } k)/N}$

x

: $w_8^0 \quad w_8^1 \quad w_8^2 \quad w_8^3 \quad w_8^4 \quad w_8^5 \quad w_8^6 \quad w_8^7$

: $w_8^0 \quad w_8^1 \quad w_8^2 \quad w_8^3 \quad -w_8^0 \quad -w_8^1 \quad -w_8^2 \quad -w_8^3$

x^2

: $w_4^0 \quad w_4^1 \quad w_4^2 \quad w_4^3 \quad w_4^0 \quad w_4^1 \quad w_4^2 \quad w_4^3$

EVALUATE in NlogN steps

p(x)

: $p_0 + p_1x^1 + p_2x^2 + p_3x^3 + p_4x^4 + p_5x^5 + p_6x^6 + p_7x^7$

split into even, odd coefficients

: $p_0 + p_2x^2 + p_4x^4 + p_6x^6 + x(p_1 + p_3x^2 + p_5x^4 + p_7x^6)$

: $p(x) = p_e(x^2) + xp_o(x^2)$

To evaluate at eighth roots of unity
use values of even, odd polynomials
at fourth roots of unity

: $p(w_8^0) = p_e(w_4^0) + w_8^0 p_o(w_4^0)$

: $p(w_8^1) = p_e(w_4^1) + w_8^1 p_o(w_4^1)$

: $p(w_8^2) = p_e(w_4^2) + w_8^2 p_o(w_4^2)$

: $p(w_8^3) = p_e(w_4^3) + w_8^3 p_o(w_4^3)$

: $p(w_8^4) = p_e(w_4^0) + w_8^4 p_o(w_4^0)$

: $p(w_8^5) = p_e(w_4^1) + w_8^5 p_o(w_4^1)$

: $p(w_8^6) = p_e(w_4^2) + w_8^6 p_o(w_4^2)$

: $p(w_8^7) = p_e(w_4^3) + w_8^7 p_o(w_4^3)$

multiplication count

$m(0) = 1$

$m(N) = 2\,m(N/2) + N$

$m(N) = N \lg N$

EVALUATE implementation

Use the perfect shuffle

$$: \quad p_0 \quad p_1 \quad p_2 \quad p_3 \quad p_4 \quad p_5 \quad p_6 \quad p_7$$

$$: \quad p_0 \quad p_2 \quad p_4 \quad p_6 \quad p_1 \quad p_3 \quad p_5 \quad p_7$$

To evaluate inplace

- unshuffle
- evaluate halves inplace recursively
- use values to compute result

Interpolating at roots of unity

For the Nth roots of unity, it turns out
that INTERPOLATE is truly the inverse
of EVALUATE

r(x)

: $r_0 + r_1x^1 + r_2x^2 + r_3x^3 + r_4x^4 + r_5x^5 + r_6x^6 + r_7x^7$

s(x)

: $s_0 + s_1x^1 + s_2x^2 + s_3x^3 + s_4x^4 + s_5x^5 + s_6x^6 + s_7x^7$

: $r(w_8^0) = s_0$	$s(w_8^0) = r_0$
: $r(w_8^1) = s_1$	$s(w_8^{-1}) = r_1$
: $r(w_8^2) = s_2$	$s(w_8^{-2}) = r_2$
: $r(w_8^3) = s_3$	$s(w_8^{-3}) = r_3$
: $r(w_8^4) = s_4$	$s(w_8^{-4}) = r_4$
: $r(w_8^5) = s_5$	$s(w_8^{-5}) = r_5$
: $r(w_8^6) = s_6$	$s(w_8^{-6}) = r_6$
: $r(w_8^7) = s_7$	$s(w_8^{-7}) = r_7$

Fundamental property (Fourier inversion theorem)

To INTERPOLATE at roots of unity,
use the EVALUATE routine (!)

Fast Fourier transform

To multiply to polynomials of degree N-1

EVALUATE both at (2N-1) roots of unity
MULTIPLY (2N-1) results term-by-term
INTERPOLATE by evaluating at (2N-1)
    inverse roots of unity

Total time proportional to N lg N
Shuffling keeps space proportional to N

Bottom line:
- can multiply numbers with millions of digits
- "multiply" = "convolution"
- basic mathematical operation on series
- basic operation in digital signal processing
    - allows "real time" processing