Multiplying Faster

On the first HW you analyzed our usual algorithm for multiplying numbers
\[ \Theta(n^2) \text{ time} \]

We can do better!
We'll describe the basic ideas by multiplying polynomials rather than integers
Advantage is we don't get confused by worrying about carries at first

Note on Polynomials
These are just formal sequences of coefficients so when we show something multiplied by \( x^k \) it just means shifted \( k \) places to the left

Polynomial Multiplication
Given:
- Degree \( m-1 \) polynomials \( P \) and \( Q \)
  \[ P = a_0 + a_1 x + a_2 x^2 + \ldots + a_{m-1} x^{m-1} \]
  \[ Q = b_0 + b_1 x + b_2 x^2 + \ldots + b_{m-1} x^{m-1} \]
Compute:
- Degree \( 2m-2 \) Polynomial \( P \cdot Q \)

Naive Divide and Conquer
Assume \( m=2k \)
- \( P = (a_0 + a_1 x + a_2 x^2 + \ldots + a_k x^k) + (a_k + a_{k+1} x + \ldots + a_{m-1} x^{m-1}) x^k \)
- \( Q = Q_0 + Q_1 x^k \)
- \( P \cdot Q = (P_0 + P_1 x^k)(Q_0 + Q_1 x^k) \)
  \[ = P_0 Q_0 + (P_0 Q_1 + P_1 Q_0) x^k + P_1 Q_1 x^{2k} \]
4 sub-problems of size \( k=m/2 \) plus linear combining
- \( T(m)=4T(m/2)+cm \)
- Solution \( T(m) = O(m^2) \)

Karatsuba’s Algorithm
A better way to compute the terms
Compute
- \( P_0 \cdot Q_0 \)
- \( P_1 \cdot Q_1 \)
- \( (P_0 + P_1)(Q_0 + Q_1) \) which is \( P_0 Q_0 + P_1 Q_1 + P_0 Q_1 + P_1 Q_0 \)
Then
- \( P_0 Q_1 + P_1 Q_0 = (P_0 + P_1)(Q_0 + Q_1) - P_0 Q_0 - P_1 Q_1 \)
3 sub-problems of size \( m/2 \) plus \( O(m) \) work
- \( T(m) = 3T(m/2) + cm \)
- \( T(m) = O(m^\alpha) \) where \( \alpha = \log_2 3 = 1.59... \)
Karatsuba’s Algorithm
Alternative

Compute
\[ A_0 = P_0 Q_0 \]
\[ A_1 = (P_0 + P_1)(Q_0 + Q_1) \], i.e. \[ P_0 Q_0 + P_1 Q_0 + P_0 Q_1 + P_1 Q_1 \]
\[ A_{-1} = (P_0 - P_1)(Q_0 - Q_1) \], i.e. \[ P_0 Q_0 - P_1 Q_0 - P_0 Q_1 + P_1 Q_1 \]

Then
\[ P_1 Q_1 = \frac{A_1 + A_{-1}}{2} - A_0 \]
\[ P_0 Q_1 + P_1 Q_0 = \frac{A_1 - A_{-1}}{2} \]

3 sub-problems of size \( m/2 \) plus \( O(m) \) work

\[ T(m) = 3 T(m/2) + cm \]
\[ T(m) = O(m^\alpha) \] where \( \alpha = \log_2 3 = 1.59... \)

What Karatsuba’s Algorithm did

For \( y = x^k \) we wanted to compute
\[ P(y)Q(y) = (P_0 + P_1 y)(Q_0 + Q_1 y) \]

We evaluated
\[ P(0) = P_0 \quad Q(0) = Q_0 \]
\[ P(1) = P_0 + P_1 \] and \( Q(1) = Q_0 + Q_1 \)
\[ P(-1) = P_0 - P_1 \] and \( Q(-1) = Q_0 - Q_1 \)

We multiplied \( P(0)Q(0), \ P(1)Q(1), \ P(-1)Q(-1) \)

We then used these 3 values to figure out what the degree 2 polynomial \( P(y)/Q(y) \) was

Interpolation

Given set of values at 5 points

Find unique degree 4 polynomial going through these points

Multiplying Polynomials by Evaluation & Interpolation

Any degree \( n-1 \) polynomial \( R(y) \) is determined by \( R(y_0), ..., R(y_{n-1}) \) for any \( n \) distinct \( y_0, ..., y_{n-1} \)

To compute \( PQ \) (assume degree at most \( n-1 \))

Evaluate \( P(y_0), ..., P(y_{n-1}) \)

Evaluate \( Q(y_0), ..., Q(y_{n-1}) \)

Multiply values \( P(y)Q(y) \) for \( i=0, ..., n-1 \)

Interpolate to recover \( PQ \)

Multiplying Polynomials by Evaluation & Interpolation

ordinary polynomial multiplication \( \Theta(n^2) \)

evaluation at \( y_0, ..., y_{n-1} \)

\( O(?) \)

interpolation from \( y_0, ..., y_{n-1} \)

\( O(?) \)

point-wise multiplication of numbers \( O(n) \)
Complex Numbers

\[ i^2 = -1 \]

To multiply complex numbers:
1. add angles
2. multiply lengths

\[ (a+bi)(c+di) = ac - bd + (ad + bc)i \]

\[ e^{\theta i} = \cos \theta + i \sin \theta \]

\[ e^{\phi i} = \cos \phi + i \sin \phi \]

\[ e^{(\theta + \phi) i} = \cos(\theta + \phi) + i \sin(\theta + \phi) \]

\[ \omega = e^{i \frac{2\pi}{n}} \]

Properties:
- \( \omega^n = 1 \)
- Any other \( z \) s.t. \( z^n = 1 \) has \( z = \omega^k \) for some \( k < n \).

If \( n \) is even, \( \omega^2 = \omega^{n/2} \) is a primitive \( n/2 \)-th root of 1.

\[ \omega^0 = 1 \]

\[ \omega^1 = \omega \]

\[ \omega^2 = i \]

\[ \omega^3 = \omega^2 \]

\[ \omega^4 = \omega^3 \]

\[ \omega^5 = \omega^4 \]

\[ \omega^6 = \omega^5 \]

\[ \omega^7 = \omega^6 \]

\[ \omega^8 = -1 \]

\[ \omega^9 = -\omega \]

\[ \omega^{10} = -\omega^2 \]

\[ \omega^{11} = -\omega^3 \]

\[ \omega^{12} = -\omega^4 \]

\[ \omega^{13} = -\omega^5 \]

\[ \omega^{14} = -\omega^6 \]

\[ \omega^{15} = -1 \]

\[ \omega^{16} = \omega^{12} \]

\[ \omega^{17} = \omega^8 \]

\[ \omega^{18} = \omega^4 \]

\[ \omega^{19} = \omega^2 \]

\[ \omega^{20} = 1 \]

Multiplying Polynomials by Fast Fourier Transform

\[ P = a_0, a_1, \ldots, a_{m-1} \]

\[ Q = b_0, b_1, \ldots, b_{m-1} \]

\[ P(1), Q(1) \]

\[ P(\omega), Q(\omega) \]

\[ P(\omega^2), Q(\omega^2) \]

\[ \ldots \]

\[ P(\omega^{n-1}), Q(\omega^{n-1}) \]

\[ R(1) = P(1)Q(1) \]

\[ R(\omega) = P(\omega)Q(\omega) \]

\[ R(\omega^2) = P(\omega^2)Q(\omega^2) \]

\[ \ldots \]

\[ R(\omega^{n-1}) = P(\omega^{n-1})Q(\omega^{n-1}) \]

\[ \sum_{k=0}^{n-1} c_k \]

\[ \rightarrow \]

\[ \begin{array}{c}
\text{ordinary polynomial multiplication} \\
\Theta(n^2)
\end{array} \]

\[ \begin{array}{c}
\text{point-wise multiplication of numbers} \\
O(n)
\end{array} \]

\[ \begin{array}{c}
\text{evaluation at } 1, \omega, \ldots, \omega^{n-1} \\
O(n \log n)
\end{array} \]

\[ \begin{array}{c}
\text{interpolation from } 1, \omega, \ldots, \omega^{n-1} \\
O(n \log n)
\end{array} \]

The key idea (since \( n \) is even)

- Also \( P_{\text{even}} \) and \( P_{\text{odd}} \) have degree \( n/2 \)
- \( P(\omega) = P_{\text{even}}(\omega^2) + \omega P_{\text{odd}}(\omega^2) \)
- \( P(-\omega) = P_{\text{even}}(\omega^2) - \omega P_{\text{odd}}(\omega^2) \)

Recursive Algorithm

- Evaluate \( P_{\text{even}} \) at \( 1, \omega^2, \omega^4, \ldots, \omega^{n/2-1} \)
- Evaluate \( P_{\text{odd}} \) at \( 1, \omega^2, \omega^4, \ldots, \omega^{n/2-1} \)
- Combine to compute \( P \) at \( 1, \omega, \omega^2, \ldots, \omega^{n-1} \)
- Combine to compute \( P \) at \( -1, -\omega, -\omega^2, \ldots, -\omega^{n/2-1} \) (i.e. \( \omega^2, \omega^4, \omega^6, \ldots, \omega^{n-1} \))

Analysis and more

- Run-time
  - \( T(n) = 2T(n/2) + cn \) so \( T(n) = O(n \log n) \)
- So much for evaluation ... what about interpolation?
  - Given \( r_0 = R(1), r_1 = R(\omega), r_2 = R(\omega^2), \ldots, r_{n-1} = R(\omega^{n-1}) \)
  - Compute \( c_0, c_1, \ldots, c_{n-1} \) s.t. \( R(x) = c_0 + c_1 x + \ldots + c_{n-1} x^{n-1} \)
Interpolation \approx \text{Evaluation: strange but true}

- **Weird fact:**
  - If we define a new polynomial
    \[ T(x) = r_0 + r_1 x + r_2 x^2 + \ldots + r_{n-1} x^{n-1} \]
    where \( r_i \) are the evaluations of \( R \) at \( 1, \omega, \ldots, \omega^{n-1} \)
  - Then \( c_k = T(\omega^k)/n \) for \( k = 0, \ldots, n-1 \)

- **So...**
  - Evaluate \( T \) at \( 1, \omega^{-1}, \omega^{-2}, \ldots, \omega^{-(n-1)} \) then divide each by \( n \)
    to get the \( c_0, \ldots, c_{n-1} \)
  - \( \omega^{-k} \) behaves just like \( \omega \) did so the same \( O(n \log n) \)
    evaluation algorithm applies!

Why this is called the discrete Fourier transform

- **Real Fourier series**
  - Given a real valued function \( f \) defined on \([0,2\pi]\)
    the Fourier series for \( f \) is given by
    \[ f(x) = a_0 + a_1 \cos(x) + a_2 \cos(2x) + \ldots + a_m \cos(mx) + \ldots \]
    where
    \[ a_m = \frac{1}{2\pi} \int_0^{2\pi} f(x) \cos(mx) \, dx \]
    is the component of \( f \) of frequency \( m \)
  - In signal processing and data compression one ignores all but the components with large \( a_m \) and
    there aren't many since \( \sum a_m = 1 \)

- **Complex Fourier series**
  - Given a function \( f \) defined on \([0,2\pi]\)
    the complex Fourier series for \( f \) is given by
    \[ f(z) = b_0 + b_1 e^{iz} + b_2 e^{2iz} + \ldots + b_m e^{miz} + \ldots \]
    where
    \[ b_m = \frac{1}{2\pi} \int_0^{2\pi} f(z) e^{-imz} \, dz \]
    is the component of \( f \) of frequency \( m \)
  - If we discretize this integral using values at \( n \) equally spaced points between 0 and \( 2\pi \) we get
    \[ b_m = \frac{1}{n} \sum_{k=0}^{n-1} f(2\pi k/n) e^{-im\pi/n} \]
    just like interpolation!