Chapter 23

Fast Fourier Transform

NEW CS 473: Theory II, Fall 2015
November 17, 2015

23.1 Introduction

23.1.0.1 Polynomials and point value pairs

Some polynomials of degree two, passing through two fixed points

\[
\begin{align*}
4 + \frac{x}{2} - \frac{5}{2}x^2 \\
3 + \frac{x}{2} - \frac{3}{2}x^2 \\
2 + \frac{x}{2} - \frac{1}{2}x^2 \\
\frac{1}{2} + \frac{x}{2} + 2x^2 \\
\frac{3}{2} + \frac{x}{2} \\
-\frac{1}{2} + \frac{x}{2} + 2x^2 \\
-2 + \frac{x}{2} + \frac{7}{2}x^2 \\
-1 + \frac{x}{2} + \frac{5}{2}x^2
\end{align*}
\]
23.1.0.2 Multiplying polynomials quickly

Definition 23.1.1. **Polynomial** \( p(x) \) of degree \( n \): a function \( p(x) = \sum_{j=0}^{n} a_j x^j = a_0 + x(a_1 + x(a_2 + \ldots + x a_n)) \).

\( x_0 \): \( p(x_0) \) can be computed in \( O(n) \) time.

“dual” (and equivalent) representation...

Theorem 23.1.2. For any set \( \{ (x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1}) \} \) of \( n \) point-value pairs such that all the \( x_k \) values are distinct, there is a unique polynomial \( p(x) \) of degree \( n - 1 \), such that \( y_k = p(x_k) \), for \( k = 0, \ldots, n - 1 \).

23.1.0.3 Polynomial via point-value

\( \{ (x_0, y_0), (x_1, y_1), (x_2, y_2) \} \): polynomial through points:

\[
p(x) = y_0 \frac{(x - x_0)(x - x_1)(x - x_2)}{(x_0 - x_0)(x_0 - x_1)(x_0 - x_2)} + y_1 \frac{(x - x_0)(x - x_1)(x - x_2)}{(x_1 - x_0)(x_1 - x_1)(x_1 - x_2)} + y_2 \frac{(x - x_0)(x - x_1)(x - x_2)}{(x_2 - x_0)(x_2 - x_1)(x_2 - x_2)}
\]

23.1.0.4 Polynomial via point-value

\( \{ (x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1}) \} \): polynomial through points:

\[
p(x) = y_0 \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} + y_1 \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} + y_2 \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}
\]

23.1.1 Polynomials: regular vs. point-value pair representation

23.1.1.1 Just because.

(A) Given \( n \) point-value pairs. Can compute \( p(x) \) in \( O(n^2) \) time.

(B) Point-value pairs representation: Multiply polynomials quickly!
(C) \( p, q \) polynomial of degree \( n - 1 \), both represented by \( 2n \) point-value pairs
\[
\{(x_0, y_0), (x_1, y_1), \ldots, (x_{2n-1}, y_{2n-1})\} \text{ for } p(x),
\text{ and } \{(x_0, y'_0), (x_1, y'_1), \ldots, (x_{2n-1}, y'_{2n-1})\} \text{ for } q(x).
\]

(D) \( r(x) = p(x)q(x) \): product.

### 23.1.2 Polynomials: regular vs. point-value pair representation

#### 23.1.2.1 Just because.

(A) In point-value representation of \( r(x) \) is
\[
\{(x_0, r(x_0)), \ldots, (x_{2n-1}, r(x_{2n-1}))\}
\]
\[
= \left\{(x_0, p(x_0)q(x_0)), \ldots, (x_{2n-1}, p(x_{2n-1})q(x_{2n-1}))\right\}
\]
\[
= \left\{(x_0, y_0y'_0), \ldots, (x_{2n-1}, y_{2n-1}y'_{2n-1})\right\}.
\]

#### 23.1.2.2 Which implies...

(A) \( p(x) \) and \( q(x) \): point-value pairs \( \implies \) compute \( r(x) = p(x)q(x) \) in linear time!
(B) ...but \( r(x) \) is in point-value representation. Bummer.
(C) ...but we can compute \( r(x) \) from this representation.
(D) Purpose: Translate quickly (i.e., \( O(n \log n) \) time) from the standard \( r \) to point-value pairs representation of polynomials.
(E) ...and back!
(F) \( \implies \) computing product of two polynomials in \( O(n \log n) \) time.
(G) **Fast Fourier Transform** is a way to do this.
(H) choosing the \( x_i \) values carefully, and using divide and conquer.

### 23.2 Computing a polynomial quickly on \( n \) values

#### 23.2.1 Computing a polynomial quickly on \( n \) values

##### 23.2.1.1 Lets just use some magic.

(A) Assume: polynomials have degree \( n - 1 \), where \( n = 2^k \).
(B) .. pad polynomials with terms having zero coefficients.
(C) **Magic set** of numbers: \( \Psi = \{x_1, \ldots, x_n\} \).

Property: \( \left| \text{SQ}(\Psi) \right| = n/2 \), where \( \text{SQ}(\Psi) = \{x^2 \mid x \in \Psi\} \).

(D) \( \left| \text{square}(\Psi) \right| = \left| \Psi \right| / 2 \).

(E) Easy to find such set...

(F) **Magic**: Have this property repeatedly...

\( \text{SQ}(\text{SQ}(\Psi)) \) has \( n/4 \) distinct values.

(G) \( \text{SQ}(\text{SQ}(\text{SQ}(\Psi))) \) has \( n/8 \) values.

(H) \( \text{SQ}'(\Psi) \) has \( n/2^i \) distinct values.

(I) Oops: No such set of real numbers.

(J) **NO SUCH SET**.
23.2.2 Collapsible sets

23.2.2.1 Assume magic...

Let us for the time being ignore this technicality, and fly, for a moment, into the land of fantasy, and assume that we do have such a set of numbers, so that $|SQ^i(\Psi)| = n/2^i$ numbers, for $i = 0, \ldots, k$. Let us call such a set of numbers **collapsible**.

23.2.3 Breaking the input polynomial into...

23.2.3.1 ... two polynomials of half the degree

(A) For a set $X = \{x_0, \ldots, x_n\}$ and polynomial $p(x)$, let

$$p(X) = \langle (x_0, p(x_0)), \ldots, (x_n, p(x_n)) \rangle.$$  

(B) $p(x) = \sum_{i=0}^{n-1} a_i x^i$ as $p(x) = u(x^2) + x \cdot v(x^2)$, where

$$u(y) = \sum_{i=0}^{n/2-1} a_{2i} y^i \quad \text{and} \quad v(y) = \sum_{i=0}^{n/2-1} a_{1+2i} y^i.$$  

(C) all even degree terms in $u(y)$, all odd degree terms in $v(y)$.

(D) maximum degree of $u(y)$, $v(y)$ is $n/2$.

23.2.3.2 FFT: The dividing stage

(A) $p(x) = \sum_{i=0}^{n-1} a_i x^i$ as $p(x) = u(x^2) + x \cdot v(x^2)$.

(B) $\Psi$: collapsible set of size $n$.

(C) $p(\Psi)$: compute polynomial of degree $n - 1$ on $n$ values.

(D) Decompose:

$$u(y) = \sum_{i=0}^{n/2-1} a_{2i} y^i \quad \text{and} \quad v(y) = \sum_{i=0}^{n/2-1} a_{1+2i} y^i.$$  

(E) Need to compute $u(x^2)$, for all $x \in \Psi$.

(F) Need to compute $v(x^2)$, for all $x \in \Psi$.

(G) $SQ(\Psi) = \{x^2 \mid x \in \Psi\}$.

(H) $\Rightarrow$ Need to compute $u(SQ(\Psi)), v(SQ(\Psi))$.

(I) $u(SQ(\Psi)), v(SQ(\Psi))$: comp. poly. degree $n/2 - 1$ on $n/2$ values.

23.2.3.3 FFT: The conquering stage

(A) $\Psi$: Collapsible set of size $n$.

(B) $p(x) = \sum_{i=0}^{n-1} a_i x^i$ as $p(x) = u(x^2) + x \cdot v(x^2)$.

(C) $u(y) = \sum_{i=0}^{n/2-1} a_{2i} y^i \quad \text{and} \quad v(y) = \sum_{i=0}^{n/2-1} a_{1+2i} y^i$.

(D) $u(SQ(\Psi)), v(SQ(\Psi))$: Computed recursively.

(E) Need to compute $p(\Psi)$.

(F) For $x \in \Psi$: Compute $p(x) = u(x^2) + x \cdot v(x^2)$.

(G) Takes constant time per single element $x \in \Psi$.

(H) Takes $O(n)$ time overall.
23.2.4 FFT algorithm

```
FFTAlg(p, X)// X: A collapsible set of n elements.
input: p(x): polynomial deg. n: \( p(x) = \sum_{i=0}^{n-1} a_i x^i \)
output: p(X)
\( u(y) = \sum_{i=0}^{n/2-1} a_{2i} y^i \)
\( v(y) = \sum_{i=0}^{n/2-1} a_{1+2i} y^i \).
Y = SQ(X) = \{ x^2 \mid x \in X \}.
U = FFTAlg(u, Y) // U = u(Y)
V = FFTAlg(v, Y) // V = v(Y)
Out \leftarrow \emptyset
for x \in X do // p(x) = u(x^2) + x \cdot v(x^2)
(x, p(x)) \leftarrow (x, U[x^2] + x \cdot V[x^2]) // U[x^2] \equiv u(x^2)
Out \leftarrow Out \cup \{ (x, p(x)) \}
return Out
```

23.2.4 Running time analysis...

23.2.4.1 ...an old foe emerges once again to serve

(A) \( T(m, n) \): Time of computing a polynomial of degree \( m \) on \( n \) values.

(B) We have that:

\[
T(n - 1, n) = 2T(n/2 - 1, n/2) + O(n).
\]

(C) The solution to this recurrence is \( O(n \log n) \).

23.2.5 Generating Collapsible Sets

23.2.5.1 Generating Collapsible Sets

(A) How to generate collapsible sets?

(B) Trick: Use complex numbers!

23.2.5.2 Complex numbers – a quick reminder

(A) Complex number: pair \((\alpha, \beta)\) of real numbers.

Written as \( \tau = \alpha + \beta i \).

(B) \( \alpha \): real part,

\( \beta \): imaginary part.

(C) \( i \) is the root of \(-1\).

(D) Geometrically: a point in the complex plane:
23.2.5.3 A useful formula: \( \cos \phi + i \sin \phi = e^{i\phi} \)

(A) By Taylor’s expansion:

\[
\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots ,
\]
\[
\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots ,
\]
and \( e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots . \)

(B) Since \( i^2 = -1: \)

\[
e^{ix} = 1 + \frac{i x}{1!} - \frac{x^2}{2!} - \frac{i x^3}{3!} + \frac{x^4}{4!} + \frac{i x^5}{5!} - \frac{x^6}{6!} \cdots = \cos x + i \sin x .
\]

23.2.5.4 Back to polar form

(A) \textbf{polar form}: \( \tau = r \cos \phi + i r \sin \phi = r(\cos \phi + i \sin \phi) = re^{i\phi} , \)
(B) \( \tau = re^{i\phi} , \quad \tau' = r'e^{i\phi'}: \) complex numbers.
(C) \( \tau \cdot \tau' = re^{i\phi} \cdot r'e^{i\phi'} = rr'e^{i(\phi+\phi')} . \)
(D) \( e^{i\phi} \) is \( 2\pi \) periodic (i.e., \( e^{i\phi} = e^{i(\phi+2\pi)} \)), and \( 1 = e^{i0} . \)
(E) \( n \)th root of 1: complex number \( \tau \) – raise it to power \( n \) get 1.
(F) \( \tau = re^{i\phi} , \) such that \( \tau^n = r^n e^{i n \phi} = e^{i0} . \)
(G) \( \implies \) \( r = 1 , \) and there must be an integer \( j , \) such that

\[
n\phi = 0 + 2\pi j \implies \phi = j(2\pi/n) .
\]

23.2.6 Roots of unity

23.2.6.1 The desire to avoid war?

For \( j = 0 , \ldots , n - 1 , \) we get the \( n \) distinct \textbf{roots of unity}.
23.2.6.2 Back to collapsible sets

(A) Can do all basic calculations on complex numbers in \( O(1) \) time.
(B) Idea: Work over the complex numbers.
(C) Use roots of unity!
(D) \( \gamma \): \( n \)th root of unity. There are \( n \) such roots, and let \( \gamma_j(n) \) denote the \( j \)th root.

\[
\gamma_j(n) = \cos((2\pi j)/n) + i\sin((2\pi j)/n) = \gamma^j.
\]

Let \( \mathcal{A}(n) = \{\gamma_0(n), \ldots, \gamma_{n-1}(n)\} \).
(E) \( |\text{SQ}(\mathcal{A}(n))| \) has \( n/2 \) entries.
(F) \( \text{SQ}(\mathcal{A}(n)) = \mathcal{A}(n/2) \)
(G) \( n \) to be a power of 2, then \( \mathcal{A}(n) \) is the required collapsible set.

23.2.6.3 The first result...

Theorem 23.2.1. Given polynomial \( p(x) \) of degree \( n \), where \( n \) is a power of two, then we can compute \( p(X) \) in \( O(n \log n) \) time, where \( X = \mathcal{A}(n) \) is the set of \( n \) different powers of the \( n \)th root of unity over the complex numbers.

23.2.7 Problem...

23.2.7.1 We can go, but can we come back?

(A) Can multiply two polynomials quickly
(B) by transforming them to the point-value pairs representation...
(C) over the \( n \)th roots of unity.
(D) Q: How to transform this representation back to the regular representation.
(E) A: Do some confusing math...

23.3 Recovering the polynomial

23.3.0.1 Recovering the polynomial

Think about FFT as a matrix multiplication operator.

\( p(x) = \sum_{i=0}^{n-1} a_i x^i \). Evaluating \( p(\cdot) \) on \( \mathcal{A}(n) \):

\[
\begin{pmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_{n-1}
\end{pmatrix}
= 
\begin{pmatrix}
1 & \gamma_0 & \gamma_0^2 & \gamma_0^3 & \cdots & \gamma_0^{n-1} \\
1 & \gamma_1 & \gamma_1^2 & \gamma_1^3 & \cdots & \gamma_1^{n-1} \\
1 & \gamma_2 & \gamma_2^2 & \gamma_2^3 & \cdots & \gamma_2^{n-1} \\
1 & \gamma_3 & \gamma_3^2 & \gamma_3^3 & \cdots & \gamma_3^{n-1} \\
\vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\
1 & \gamma_{n-1} & \gamma_{n-1}^2 & \gamma_{n-1}^3 & \cdots & \gamma_{n-1}^{n-1}
\end{pmatrix}
\begin{pmatrix}
a_0 \\
a_1 \\
a_2 \\
a_3 \\
\vdots \\
a_{n-1}
\end{pmatrix},
\]

where \( \gamma_j = \gamma_j(n) = (\gamma_1(n))^j \) is the \( j \)th power of the \( n \)th root of unity, and \( y_j = p(\gamma_j) \).
23.3.1 The Vandermonde matrix

23.3.1.1 Because every matrix needs a name

$V$ is the **Vandermonde** matrix.

$V^{-1}$: inverse matrix of $V$

Vandermonde matrix. And let multiply the above formula from the left. We get:

\[
\begin{pmatrix}
  y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_{n-1}
\end{pmatrix} = V \begin{pmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_{n-1}
\end{pmatrix} \implies \begin{pmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_{n-1}
\end{pmatrix} = V^{-1} \begin{pmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_{n-1}
\end{pmatrix}.
\]

23.3.2 The inverse Vandermonde matrix

23.3.2.1 ..for the rescue

(A) Recover the polynomial $p(x)$ from the point-value pairs \[ \{(γ_0, p(γ_0)), (γ_1, p(γ_1)), \ldots, (γ_{n-1}, p(γ_{n-1}))\} \]

(B) by doing a single matrix multiplication of $V^{-1}$ by the vector \[ [y_0, y_1, \ldots, y_{n-1}] \].

(C) Multiplying a vector with $n$ entries with $n \times n$ matrix takes $O(n^2)$ time.

(D) No benefit so far...

23.3.3 What is the inverse of the Vandermonde matrix

23.3.3.1 Vandermonde matrix is famous, beautiful and well known – a celebrity matrix

Claim 23.3.1.

\[
V^{-1} = \frac{1}{n} \begin{pmatrix}
  1 & β_0 & β_0^2 & β_0^3 & \ldots & β_0^{n-1} \\
  1 & β_1 & β_1^2 & β_1^3 & \ldots & β_1^{n-1} \\
  1 & β_2 & β_2^2 & β_2^3 & \ldots & β_2^{n-1} \\
  \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
  1 & β_{n-1} & β_{n-1}^2 & β_{n-1}^3 & \ldots & β_{n-1}^{n-1}
\end{pmatrix},
\]

where $β_j = (γ_j(n))^{-1}$.

23.3.3.2 Proof

Consider the $(u, v)$ entry in the matrix $C = V^{-1}V$. We have

\[
C_{u,v} = \sum_{j=0}^{n-1} \frac{(β_u)^j(γ_j)^v}{n}.
\]
As \( \gamma_j = (\gamma^1)^j \). Thus,

\[
C_{u,v} = \sum_{j=0}^{n-1} \left( \frac{\beta_u}{n} \right)^j ((\gamma^1)^v)^j = \sum_{j=0}^{n-1} \frac{\beta_u^j (\gamma^1)^v^j}{n} = \sum_{j=0}^{n-1} \frac{(\beta_u \gamma^v)^j}{n}.
\]

Clearly, if \( u = v \) then

\[
C_{u,u} = \frac{1}{n} \sum_{j=0}^{n-1} (\beta_u)^j = \frac{1}{n} \sum_{j=0}^{n-1} (1)^j = \frac{n}{n} = 1.
\]

### 23.3.3.3 Proof continued...

If \( u \neq v \) then,

\[
\beta_u \gamma_v = (\gamma_u)^{-1} \gamma_v = (\gamma^1)^{-u} \gamma^v = (\gamma^1)^{v-u} = \gamma^{v-u}.
\]

And

\[
C_{u,v} = \frac{1}{n} \sum_{j=0}^{n-1} (\gamma_v)^j = \frac{1}{n} \cdot \frac{\gamma_v^n - 1}{\gamma_v - 1} = \frac{1}{n} \cdot \frac{1 - 1}{\gamma_v - 1} = 0,
\]

Proved that the matrix \( C \) have ones on the diagonal and zero everywhere else.

### 23.3.3.4 Recap...

(A) \( n \) point-value pairs \( \{(\gamma_0, y_0), \ldots, (\gamma_{n-1}, y_{n-1})\} \): of polynomial \( p(x) = \sum_{i=0}^{n-1} a_i x^i \) over \( n \)th roots of unity.

(B) Recover coefficients of polynomial by multiplying \([y_0, y_1, \ldots, y_n]\) by \( V^{-1} \):

\[
\begin{pmatrix}
  a_0 \\
  a_1 \\
  a_2 \\
  \vdots \\
  a_{n-1}
\end{pmatrix} = \frac{1}{n} \begin{pmatrix}
  1 & \beta_0 & \beta_0^2 & \beta_0^3 & \cdots & \beta_0^{n-1} \\
  1 & \beta_1 & \beta_1^2 & \beta_1^3 & \cdots & \beta_1^{n-1} \\
  1 & \beta_2 & \beta_2^2 & \beta_2^3 & \cdots & \beta_2^{n-1} \\
  \vdots \\
  1 & \beta_{n-1} & \beta_{n-1}^2 & \beta_{n-1}^3 & \cdots & \beta_{n-1}^{n-1}
\end{pmatrix} \begin{pmatrix}
  y_0 \\
  y_1 \\
  y_2 \\
  \vdots \\
  y_{n-1}
\end{pmatrix}.
\]

(C) \( W(x) = \sum_{i=0}^{n-1} (y_i/n) x^i \): \( a_i = W(\beta_i) \).

### 23.3.3.5 Recovering continued...

(A) recover coefficients of \( p(\cdot) \).

(B) \( \cdots \) compute \( W(\cdot) \) on \( n \) values: \( \beta_0, \ldots, \beta_{n-1} \).

(C) \( \{\beta_0, \ldots, \beta_{n-1}\} = \{\gamma_0, \ldots, \gamma_{n-1}\} \).

(D) Indeed \( \beta_i^n = (\gamma_i^{-1})^n = (\gamma_i^n)^{-1} = 1^{-1} = 1 \).

(E) Apply the FFTAlg algorithm on \( W(x) \) to compute \( a_0, \ldots, a_{n-1} \).
23.3.3.6 Result

Theorem 23.3.2. Given \( n \) point-value pairs of a polynomial \( p(x) \) of degree \( n - 1 \) over the set of \( n \) powers of the \( n \)th roots of unity, we can recover the polynomial \( p(x) \) in \( O(n \log n) \) time.

Theorem 23.3.3. Given two polynomials of degree \( n \), they can be multiplied in \( O(n \log n) \) time.

23.4 Convolutions

23.4.0.1 Convolutions

(A) Two vectors: \( A = [a_0, a_1, \ldots, a_n] \) and \( B = [b_0, \ldots, b_n] \).

(B) dot product \( A \cdot B = \langle A, B \rangle = \sum_{i=0}^{n} a_i b_i \).

(C) \( A_r \): shifting of \( A \) by \( n - r \) locations to the left (D) Padded with zeros: \( a_j = 0 \) for \( j \notin \{0, \ldots, n\} \).

(E) \( A_r = [a_{n-r}, a_{n+1-r}, a_{n+2-r}, \ldots, a_{2n-r}] \) where \( a_j = 0 \) if \( j \notin [0, \ldots, n] \).

(F) Observation: \( A_n = A \).

23.4.0.2 Example of shifting

Example 23.4.1. For \( A = [3, 7, 9, 15] \), \( n = 3 \)

\[ A_2 = [7, 9, 15, 0], \]

\[ A_5 = [0, 0, 3, 7]. \]

23.4.0.3 Definition

Definition 23.4.2. Let \( c_i = A_i \cdot B = \sum_{j=n-i}^{2n-i} a_j b_{j-n+i} \), for \( i = 0, \ldots, 2n \). The vector \([c_0, \ldots, c_{2n}]\) is the convolution of \( A \) and \( B \).

question How to compute the convolution of two vectors of length \( n \)?

23.4.0.4 Convolution via multiplication polynomials

(A) \( p(x) = \sum_{i=0}^{n} \alpha_i x^i \), and \( q(x) = \sum_{i=0}^{n} \beta_i x^i \).

(B) Coefficient of \( x^i \) in \( r(x) = p(x)q(x) \) is \( d_i = \sum_{j=0}^{i} \alpha_j \beta_{i-j} \).

(C) Want to compute \( c_i = A_i \cdot B = \sum_{j=n-i}^{2n-i} a_j b_{j-n+i} \).

(D) Set \( \alpha_i = a_i \) and \( \beta_j = b_{n-j} \).

23.4.0.5 Convolution by example

(A) Consider coefficient of \( x^2 \) in product of \( p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 \) and \( q(x) = b_0 + b_1 x + b_2 x^2 + b_3 x^3 \).

(B) Sum of the entries on the anti diagonal:

<table>
<thead>
<tr>
<th>( b_0 )</th>
<th>( a_0 + )</th>
<th>( a_1 x )</th>
<th>(+a_2 x^2)</th>
<th>(+a_3 x^3)</th>
</tr>
</thead>
</table>
| \(+b_1 x\) | \( a_0 b_1 x^2\) | \( a_1 b_1 x^2\) | \( a_2 b_0 x^2\)
| \(+b_2 x^2\) | \( a_0 b_2 x^2\) | \( a_1 b_2 x^2\) | \( a_3 b_0 x^2\)
| \(+b_3 x^3\) | \( a_0 b_3 x^3\) | \( a_1 b_3 x^3\) | \( a_2 b_3 x^3\)

(C) entry in the \( i \)th row and \( j \)th column is \( a_i b_j \).
23.4.0.6 Convolution

Theorem 23.4.3. Given two vectors $A = [a_0, a_1, \ldots, a_n]$, $B = [b_0, \ldots, b_n]$ one can compute their convolution in $O(n \log n)$ time.

Proof: Let $p(x) = \sum_{i=0}^{n} a_{n-i} x^i$ and let $q(x) = \sum_{i=0}^{n} b_i x^i$. Compute $r(x) = p(x)q(x)$ in $O(n \log n)$ time using the convolution theorem. Let $c_0, \ldots, c_{2n}$ be the coefficients of $r(x)$. It is easy to verify, as described above, that $[c_0, \ldots, c_{2n}]$ is the convolution of $A$ and $B$. \hfill \blacksquare