Chapter 3

Convolutions and the Discrete Fourier Transform

3.1 The Cooley-Tukey Algorithm

Let $n = pq$, with $p, q > 1$. Given a vector $\boldsymbol{x}$ of size n, the *Cooley-Tukey algorithm* computes $DFT_n(\boldsymbol{x}) = F_n\boldsymbol{x}$ in terms of the lower-order transforms DFT_p and DFT_q by performing the following five steps:

1. Arrange $\boldsymbol{x}$ into a $p \times q$ matrix X, in row major order;
2. For $0 \leq j \leq q-1$, substitute column X^j of matrix X with $DFT_p(X^j) = F_pX^j$;
3. For $0 \leq i \leq p-1$, $0 \leq j \leq q-1$, multiply the (i,j)-th entry of the matrix by the *twiddle factor* ω_n^{ij};
4. For $0 \leq i \leq p-1$ substitute row X_i of matrix X with $DFT_q(X_i) = F_qX_i$;
5. Read out $\boldsymbol{y} = DFT_n(\boldsymbol{x})$ by enumerating the entries of the resulting matrix in column-major order.

For $n = 2^k$, the Cooley-Tukey algorithm yields the FFT($\boldsymbol{x}$) algorithm seen in class. Indeed, let $n = p \times q$, with $p = n/2$ and $q = 2$. Then, the first step creates the following

matrix with $n/2$ rows and 2 columns:

$$X = \begin{pmatrix} x_0 & x_1 \\ \vdots & \vdots \\ x_{2i} & x_{2i+1} \\ \vdots & \vdots \\ x_{n-2} & x_{n-1} \end{pmatrix}.$$

Observe that $X^0 = \boldsymbol{x}^{[0]}$ and $X^1 = \boldsymbol{x}^{[1]}$, in other words, the two columns separate the even-indexed entries of $\boldsymbol{x}$ from the odd-indexed entries, as in the "Divide" step of FFT. Step 2 of the Cooley-Tukey algorithm requires transforming the columns independently. Therefore, if

$$\boldsymbol{y}^{[0]} = F_{n/2}\boldsymbol{x}^{[0]} \text{ and } \boldsymbol{y}^{[1]} = F_{n/2}\boldsymbol{x}^{[1]}$$

we obtain

$$X = \begin{pmatrix} y_0^{[0]} & y_0^{[1]} \\ \vdots & \vdots \\ y_i^{[0]} & y_i^{[1]} \\ \vdots & \vdots \\ y_{n/2-1}^{[0]} & y_{n/2-1}^{[1]} \end{pmatrix},$$

which corresponds to the "Recurse" step of FFT. In Step 3 (multiplication by the twiddle factors), the i-th component of the second column X^1 is multiplied by $\omega_n^{i \cdot 1} = \omega_n^i$, yielding

$$X = \begin{pmatrix} y_0^{[0]} & {\omega_n}^0 y_1^{[1]} \\ \vdots & \vdots \\ y_i^{[0]} & {\omega_n}^i y_i^{[1]} \\ \vdots & \vdots \\ y_{n/2-1}^{[0]} & {\omega_n}^{n/2-1} y_{n/2-1}^{[1]} \end{pmatrix}.$$

In Step 4, we transform each row $(y_i^{[0]}, {\omega_n}^i y_i^{[1]})$. Since

$$F_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},$$

the i-th row of X becomes

$$\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} y_i^{[0]} \\ {\omega_n}^i y_i^{[1]} \end{pmatrix} = \begin{pmatrix} y_i^{[0]} + {\omega_n}^i y_i^{[1]} \\ y_i^{[0]} - {\omega_n}^i y_i^{[1]} \end{pmatrix}.$$

Finally, obtaining the components of $\boldsymbol{y} = F_n \boldsymbol{x}$ in column major order amounts to say that, for $0 \leq i \leq n/2 - 1$,

$$y_i = y_i^{[0]} + {\omega_n}^i y_i^{[1]} \text{ and } y_{i+n/2} = y_i^{[0]} - {\omega_n}^i y_i^{[1]},$$

which is exactly the recombination performed in the "Conquer" step of FFT.

Exercise 3.1 Let $n = 12$, $p = 3$, $q = 4$, and let $\boldsymbol{x} = (0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1)$. Compute $DFT_{12}(\boldsymbol{x})$ by applying the Cooley-Tukey algorithm and showing the matrix at the end of each step.

Answer: Let $\omega = \omega_{12} = e^{\pi i/6} = \sqrt{3}/2 + i/2$. On input $\boldsymbol{x} = (0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1)$, the Cooley-Tukey algorithm executes as follows.

Step 1: Arrange $\boldsymbol{x}$ into a 3×4 matrix, in row major order. We obtain:

$$X = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

Step 2: Transform columns. We must replace each column X^j with $F_3 X^j$, where F_3 is based on the third root $\omega_3 = -1/2 + (\sqrt{3}/2)i$. The result is:

$$X = \begin{bmatrix} 1 & 0 & 3 & 2 \\ -\frac{1}{2} + \frac{\sqrt{3}}{2}i & 0 & 0 & \frac{1}{2} - \frac{\sqrt{3}}{2}i \\ -\frac{1}{2} - \frac{\sqrt{3}}{2}i & 0 & 0 & \frac{1}{2} + \frac{\sqrt{3}}{2}i \end{bmatrix}.$$

Step 3: Multiply entry (i, j) by ω^{ij}. Note that the first column and the first row will be left unchanged (multiplied by $\omega^0 = 1$), and there are several 0-entries in the matrix. In fact, we only need to compute $\omega^3 = e^{\pi i/2} = i$ and $\omega^6 = e^{\pi i} = -1$. The resulting matrix is:

$$X = \begin{bmatrix} 1 & 0 & 3 & 2 \\ -\frac{1}{2} + \frac{\sqrt{3}}{2}i & 0 & 0 & \frac{\sqrt{3}}{2} + \frac{1}{2}i \\ -\frac{1}{2} - \frac{\sqrt{3}}{2}i & 0 & 0 & -\frac{1}{2} - \frac{\sqrt{3}}{2}i \end{bmatrix}.$$

Step 4: Transform rows. Similarly to Step 2, we replace each row X_i with $F_4 X_i$, where F_4 is based on the fourth root $\omega_4 = i$. The result is:

$$D = \begin{bmatrix} 6 & -2-2i & 2 & -2+2i \\ \frac{\sqrt{3}-1}{2} + \frac{1+\sqrt{3}}{2}i & 0 & -\frac{1+\sqrt{3}}{2} + \frac{\sqrt{3}-1}{2}i & -1+\sqrt{3}i \\ -1-\sqrt{3}i & -\frac{1+\sqrt{3}}{2} + \frac{1-\sqrt{3}}{2}i & 0 & \frac{\sqrt{3}-1}{2} - \frac{1+\sqrt{3}}{2}i \end{bmatrix}.$$

Step 5: Read out the transform in column major order. We finally obtain:

$$\begin{aligned} &DFT_{12}(\boldsymbol{x}) = \\ &\quad \left(6, \frac{\sqrt{3}-1}{2} + \frac{1+\sqrt{3}}{2}i, -1-\sqrt{3}i, -2-2i, 0, -\frac{1+\sqrt{3}}{2} + \frac{1-\sqrt{3}}{2}i,\right. \\ &\qquad \left. 2, -\frac{1+\sqrt{3}}{2} + \frac{\sqrt{3}-1}{2}i, 0, -2+2i, -1+\sqrt{3}i, \frac{\sqrt{3}-1}{2} - \frac{1+\sqrt{3}}{2}i\right). \end{aligned}$$

□

3.2 Linear and Cyclic Convolution

Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be two arbitrary vectors of n components. The *linear convolution* of $\boldsymbol{a}$ and $\boldsymbol{b}$, denoted $\boldsymbol{w} = \boldsymbol{a} \star \boldsymbol{b}$, is a vector of $2n-1$ components such that, for $0 \leq i \leq 2n-1$,

$$w_i = \sum_{j=\max\{0,i-n+1\}}^{\min\{i,n-1\}} a_j b_{i-j}, \tag{3.1}$$

where the lower and upper bounds in the summation are chosen in such a way that the indices j and $i-j$ always range between 0 and $n-1$. Recall that w_i is the i-th coefficient of the polynomial of degree bound $2n$ which is obtained by multiplying the two polynomials whose coefficient representations are $\boldsymbol{a}$ and $\boldsymbol{b}$. By evaluating the polynomials on the $2n$ $2n$-th roots of unity, pointwise-multiplying the values and interpolating from the resulting point representation vector, we obtain the following theorem:

Theorem 3.1 (Linear Convolution Theorem) *Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be two arbitrary vectors of n components, and let $\boldsymbol{w} = \boldsymbol{a} \star \boldsymbol{b}$. Then*

$$(\boldsymbol{w}|\boldsymbol{0}_1) = DFT_{2n}^{-1}\left(DFT_{2n}(\boldsymbol{a}|\boldsymbol{0}_n) \odot DFT_{2n}(\boldsymbol{b}|\boldsymbol{0}_n)\right),$$

where $\odot$ denotes component-wise product, and $(\boldsymbol{x}|\boldsymbol{0}_k)$ denotes the vector obtained by padding

vector $\boldsymbol{x}$ with k zeroes.

Theorem (3.1) gives us a way to compute $\boldsymbol{w}$ in $\Theta(n \log n)$ time by applying the FFT algorithm.

Let us now introduce a new vector operator. For $\boldsymbol{a}$ and $\boldsymbol{b}$, vectors of n components, we define the *cyclic convolution* (also called *wrapped convolution*) of $\boldsymbol{a}$ and $\boldsymbol{b}$, denoted $\boldsymbol{a} \circledast \boldsymbol{b}$, as a vector $\boldsymbol{z}$ of n components such that, for $0 \leq i \leq n-1$,

$$z_i = \sum_{j=0}^{n-1} a_j b_{(i-j) \bmod n}. \tag{3.2}$$

Note the similarity between (3.1) and (3.2). However, recall that if $\boldsymbol{a}$ and $\boldsymbol{b}$ have n components, then $\boldsymbol{a} \star \boldsymbol{b}$ has $2n-1$ components, while $\boldsymbol{a} \circledast \boldsymbol{b}$ has only n components.

Cyclic convolution can be thought of as a "wrapped" version of linear convolution. To see this, note that, for $0 \leq i \leq n-1$,

$$z_i = \sum_{j=0}^{i} a_j b_{i-j} + \sum_{j=i+1}^{n-1} a_j b_{n+i-j} \tag{3.3}$$

(note that the second summation evaluates to zero for $i = n-1$). To obtain z_i, we start by multiplying, a_0 by b_i, then a_1 by b_{i-1} ... until we multiply a_i by b_0. Then, we proceed by "wrapping around" vector $\boldsymbol{b}$ and multiplying a_{i+1} by b_{n-1}, a_{i+2} by b_{n-2} and so forth.

Observe the multiplication pattern of the a_i's and the b_j's in cyclic convolution:

$$\begin{array}{rcccccccccc}
z_0 & = & a_0 b_0 & + & a_1 b_{n-1} & + & \ldots & + & a_{n-2} b_2 & + & a_{n-1} b_1 \\
z_1 & = & a_0 b_1 & + & a_1 b_0 & + & \ldots & + & a_{n-2} b_3 & + & a_{n-1} b_2 \\
\vdots & & & & & & & & & & \\
z_i & = & a_0 b_i & + & a_1 b_{i-1} & + & \ldots & + & a_{n-2} b_{i+2} & + & a_{n-1} b_{i+1} \\
\vdots & & & & & & & & & & \\
z_{n-1} & = & a_0 b_{n-1} & + & a_1 b_{n-2} & + & \ldots & + & a_{n-2} b_1 & + & a_{n-1} b_0
\end{array}$$

We can write the above system as $\boldsymbol{z} = C(\boldsymbol{b}) \times \boldsymbol{a}$, where

$$C(\boldsymbol{b}) = \begin{bmatrix} b_0 & b_{n-1} & \dots & b_2 & b_1 \\ b_1 & b_0 & \dots & b_3 & b_2 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ b_i & b_{i-1} & \dots & b_{i+2} & b_{i+1} \\ \vdots & \vdots & \dots & \vdots & \vdots \\ b_{n-1} & b_{n-2} & \dots & b_1 & b_0 \end{bmatrix}$$

Note that the columns of $C(\boldsymbol{b})$ are obtained as consecutive *cyclic right shifts* of $\boldsymbol{b}$. Matrix $C(\boldsymbol{b})$ is called a *circulant* matrix with first column $\boldsymbol{b}$. A circulant matrix admits a very compact representation, since the matrix is uniquely specified by its first column.

Note that computing $\boldsymbol{w} = \boldsymbol{a} \circledast \boldsymbol{b}$ according to the definition takes $\Theta(n^2)$ time. However, it turns out that we can use the FFT to compute cyclic convolutions in $\Theta(n \log n)$ time. We have:

Theorem 3.2 (Cyclic Convolution Theorem) *Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be two arbitrary vectors of n components, and let $\boldsymbol{z} = \boldsymbol{a} \circledast \boldsymbol{b}$. Then*

$$\boldsymbol{z} = DFT_n^{-1}\left(DFT_n(\boldsymbol{a}) \odot DFT_n(\boldsymbol{b})\right), \tag{3.4}$$

where $\odot$ denotes component-wise product.

Therefore, simply computing the DFT's of $\boldsymbol{a}$ and $\boldsymbol{b}$ *with no padding*, multiplying their components and then taking the inverse DFT gives us the cyclic convolution of $\boldsymbol{a}$ and $\boldsymbol{b}$.

Proof: It suffices to show that $DFT_n(\boldsymbol{z}) = DFT_n(\boldsymbol{a}) \odot DFT_n(\boldsymbol{b})$. We have:

$$(DFT_n(\boldsymbol{a}))_i = \sum_{j=0}^{n-1} a_j \omega_n^{ij} \quad \text{and} \quad (DFT_n(\boldsymbol{b}))_i = \sum_{k=0}^{n-1} b_k \omega_n^{ik},$$

therefore

$$(DFT_n(\boldsymbol{a}))_i \cdot (DFT_n(\boldsymbol{b}))_i = \left(\sum_{j=0}^{n-1} a_j \omega_n^{ij}\right)\left(\sum_{k=0}^{n-1} b_k \omega_n^{ik}\right) = \sum_{j=0}^{n-1}\sum_{k=0}^{n-1} a_j b_k \omega_n^{i(j+k)}. \tag{3.5}$$

On the other hand, from Equation (3.3) we have:

$$(DFT_n(\boldsymbol{z}))_i \;=\; \sum_{p=0}^{n-1}\left(\sum_{s=0}^{p} a_s b_{p-s} + \sum_{s=p+1}^{n-1} a_s b_{n+p-s}\right)\omega_n^{ip}$$

$$= \sum_{p=0}^{n-1} \sum_{s=0}^{p} a_s b_{p-s} \omega_n^{ip} + \sum_{p=0}^{n-1} \sum_{s=p+1}^{n-1} a_s b_{n+p-s} \omega_n^{ip}. \quad (3.6)$$

Let us now show that, for any $0 \leq i \leq n-1$, Formulae (3.5) and (3.6) coincide. For this purpose, it suffices to note that each term $a_\ell b_m$, with $0 \leq \ell, m \leq n-1$, appears only once in (3.5), multiplied by $\omega_n^{i(\ell+m)}$. In (3.6), if $\ell + m < n$, then $a_\ell b_m$ appears only once in the first double summation, for $p = \ell + m$ and $s = \ell$, and multiplied by $\omega_n^{ip} = \omega_n^{i(\ell+m)}$. If $\ell + m \geq n$, then $a_\ell b_m$ appears only once in the second double summation, for $p = \ell + m - n$ and $s = \ell$, and multiplied by $\omega_n^{ip} = \omega_n^{i(\ell+m-n)} = \omega_n^{i(\ell+m)} \omega_n^{-in} = \omega_n^{i(\ell+m)}$. The theorem follows. □

The following are immediate corollaries of Theorems 3.1 and 3.2.

Corollary 3.1 *Let* $\boldsymbol{a}$ *and* $\boldsymbol{b}$ *be two arbitrary vectors of* n *components. Then*

$$(\boldsymbol{a} \star \boldsymbol{b} | \mathbf{0}_1) = (\boldsymbol{a} | \mathbf{0}_n) \circledast (\boldsymbol{b} | \mathbf{0}_n).$$

Corollary 3.2 *Let* $\boldsymbol{a}$ *and* $\boldsymbol{b}$ *be two arbitrary vectors of* n *components. Then*

$$\boldsymbol{a} \circledast \boldsymbol{b} = \boldsymbol{b} \circledast \boldsymbol{a} = C(\boldsymbol{b}) \times \boldsymbol{a} = C(\boldsymbol{a}) \times \boldsymbol{b}$$

Corollary 3.1 gives us a way to compute $\boldsymbol{w} = \boldsymbol{a} \star \boldsymbol{b}$ through a cyclic convolution. In what follows, we investigate the inverse relation: in particular, we will write $\boldsymbol{z} = \boldsymbol{a} \circledast \boldsymbol{b}$ as a function of $\boldsymbol{w}$.

Recall that

$$\begin{aligned} z_i &= \sum_{j=0}^{i} a_j b_{i-j} + \sum_{j=i+1}^{n-1} a_j b_{n+i-j} \\ &= z_i^1 + z_i^2, \end{aligned} \quad (3.7)$$

for $0 \leq i \leq n-1$. Note that $z_{n-1}^2 = 0$. By making the bounds in the summation explicit, we can rewrite (3.1) as

$$w_i = \begin{cases} \sum_{j=0}^{i} a_j b_{i-j}, & \text{for } 0 \leq i \leq n-1, \\ \\ \sum_{j=i-n+1}^{n-1} a_j b_{i-j}, & \text{for } n \leq i \leq 2n-2. \end{cases} \quad (3.8)$$

From (3.7) and (3.8) it immediately follows that $z_{n-1}^1 = w_{n-1}$, therefore $z_{n-1} = w_{n-1}$. For

$0 \leq i \leq n-2$ we have:

$$\begin{aligned} z_i^1 &= w_i \\ \\ z_i^2 &= \sum_{j=i+1}^{n-1} a_j b_{n+i-j} \\ &= \sum_{\substack{j = (i+n)- \\ -n+1}}^{n-1} a_j b_{(i+n)-j} \\ &= w_{i+n}. \end{aligned}$$

Therefore, for $0 \leq i \leq n-2$, $z_i = w_i + w_{i+n}$.

3.3 Bluestein's Technique

Let $n \geq 1$ be an arbitrary integer (not necessarily a power of two). Given a complex vector $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$, let $\boldsymbol{y} = DFT_n(\boldsymbol{x})$, that is,

$$y_i = \sum_{k=0}^{n-1} x_k \omega_n^{ik}, \quad \text{for } 0 \leq i \leq n-1. \tag{3.9}$$

Bluestein's technique reduces the computation of $\boldsymbol{y}$ to a cyclic convolution between two vectors of size $m = \Theta(n)$, with m a power of two, where the two vectors can be easily determined in $\Theta(n)$ time. Observe that this reduction immediately yields an algorithm to compute $\boldsymbol{y} = DFT_n(\boldsymbol{x})$ in time $\Theta(n \log n)$ through the Cyclic Convolution Theorem and the FFT algorithm for vector sizes that are integer powers of two.

We proceed as follows. Let β be one of the two square roots of ω_n (e.g., take the root whose polar representation is $e^{i\pi/n}$)[1]. Since $(i-k)^2 = i^2 + k^2 - 2ik$, we have that $\omega_n^{ik} = \beta^{2ik} = \beta^{i^2}\beta^{k^2}\beta^{-(i-k)^2}$, hence we can rewrite equation (3.9) as

$$y_i \beta^{-i^2} = \sum_{k=0}^{n-1} \left(x_k \beta^{k^2}\right) \beta^{-(i-k)^2}. \tag{3.10}$$

Let us now define $a_k = x_k \beta^{k^2}$ and $b_k = \beta^{-k^2}$, for $0 \leq k \leq n-1$. Also, let $m =$

[1]As an aside, it is important to observe that taking powers or roots in the complex field, is computationally very fast when complex numbers are given in polar representation. In particular, taking powers or roots of numbers with unit modulus $\rho = 1$ amounts to perform simple arithmetic operations (products or divisions) on the second coordinate θ.

$2^{\lceil \log_2(2n-1) \rceil}$. Observe that m is the smallest power of two greater or equal to $2n-1$, hence $2n-1 \leq m \leq 2(2n-1) \leq 4n$. We can define the following two vectors of size m:

$$\begin{aligned} \boldsymbol{a'} &= (a_0, a_1, \ldots, a_{n-1} | \mathbf{0}_{m-n}) \\ \boldsymbol{b'} &= (b_0, b_1, \ldots, b_{n-1} | \mathbf{0}_{m-(2n-1)} | b_{n-1}, \ldots, b_2, b_1). \end{aligned}$$

Let $\boldsymbol{z} = \boldsymbol{a'} \circledast \boldsymbol{b'}$, and consider the first n components of $\boldsymbol{z}$. For $0 \leq i \leq n-1$ we have:

$$\begin{aligned} z_i &= \sum_{k=0}^{m-1} [\boldsymbol{a'}]_k [\boldsymbol{b'}]_{(i-k) \bmod m} \\ &= \sum_{k=0}^{n-1} a_k [\boldsymbol{b'}]_{(i-k) \bmod m} && (3.11) \\ &= \sum_{k=0}^{i} a_k b_{i-k} + \sum_{k=i+1}^{n-1} a_k b_{k-i} && (3.12) \\ &= \sum_{k=0}^{i} x_k \beta^{k^2} \beta^{-(i-k)^2} + \sum_{k=i+1}^{n-1} x_k \beta^{k^2} \beta^{-(k-i)^2} \\ &= \sum_{k=0}^{n-1} x_k \beta^{k^2} \beta^{-(i-k)^2} && (3.13) \\ &= y_i \beta^{-i^2}, && (3.14) \end{aligned}$$

where Equality (3.11) follows from the fact that $[\boldsymbol{a'}]_k = a_k$ for $0 \leq k \leq n-1$, and $[\boldsymbol{a'}]_k = 0$ for $n \leq k \leq m-1$; Equality (3.12) follows from the fact that $[\boldsymbol{b'}]_{(i-k) \bmod m} = b_{(i-k)}$, if $0 \leq i-k \leq n-1$, and $[\boldsymbol{b'}]_{(i-k) \bmod m} = [\boldsymbol{b'}]_{m+(i-k)} = b_{k-i}$, if $-(n-1) \leq i-k < 0$; Equality (3.13) holds since $\beta^{-(k-i)^2} = \beta^{-(i-k)^2}$, for any $0 \leq i, k \leq n-1$; and finally, Equality (3.14) immediately follows from Equality (3.10).

Putting it all together, in order to compute $\boldsymbol{y} = DFT_n(\boldsymbol{x})$, we first compute the vectors $\boldsymbol{a'}$ and $\boldsymbol{b'}$ in $O(n)$ time and then obtain their cyclic convolution $\boldsymbol{z} = \boldsymbol{a'} \circledast \boldsymbol{b'}$. Observe that this is a convolution between vectors of size $m = \Theta(n)$, a power of two, and can thus be computed through the Cyclic Convolution Theorem and the FFT algorithm in time $\Theta(n \log n)$. Once we have $z_i = y_i \beta^{-i^2}$, for $0 \leq i \leq n$, we can finally compute $\boldsymbol{y}$ in additional $\Theta(n)$ time as

$$y_i = z_i \beta^{i^2}, \quad \text{for } 0 \leq i \leq n-1.$$

The overall time is still $\Theta(n \log n)$.

3.4 Circulant Matrices

Given a vector $\boldsymbol{a} = (a_0, a_1, \ldots, a_{n-1})$, the *circulant* matrix $C(\boldsymbol{a})$ is an $n \times n$ matrix whose first column is $\boldsymbol{a}$, while the remaining columns are obtained as consecutive cyclic right shifts of $\boldsymbol{a}$. In this subsection we design and analyze efficient algorithms for the following problems:

(a) Determining the product of two circulant matrices.

(b) Determining a solution (if any) to the linear system $C(\boldsymbol{a})\boldsymbol{x} = \boldsymbol{b}$.

(c) Determining whether $C(\boldsymbol{a})$ is invertible and, if so, computing its inverse.

Note that

$$C(\boldsymbol{a}) = \begin{bmatrix} a_0 & a_{n-1} & \cdots & a_2 & a_1 \\ a_1 & a_0 & \cdots & a_3 & a_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n-2} & a_{n-3} & \cdots & a_0 & a_{n-1} \\ a_{n-1} & a_{n-2} & \cdots & a_1 & a_0 \end{bmatrix},$$

therefore $[C(\boldsymbol{a})]_{ij} = a_{(i-j) \bmod n}$, for $0 \leq i, j \leq n-1$.

(a) Let $C(\boldsymbol{a})$ and $C(\boldsymbol{b})$ be two circulant matrices. By the definition of row-by-column product we have, for $0 \leq i, j \leq n-1$,

$$\begin{aligned} [C(\boldsymbol{a}) \times C(\boldsymbol{b})]_{ij} &= \sum_{s=0}^{n-1} [C(\boldsymbol{a})]_{is} [C(\boldsymbol{b})]_{sj} \\ &= \sum_{s=0}^{n-1} a_{(i-s) \bmod n} b_{(s-j) \bmod n}. \end{aligned}$$

Let $k = (i-s) \bmod n$. Note that when s varies between 0 and $n-1$, so does k. Moreover, $s = (i-k) \bmod n$. By substituting s with k in the above summation we obtain:

$$\begin{aligned} [C(\boldsymbol{a}) \times C(\boldsymbol{b})]_{ij} &= \sum_{k=0}^{n-1} a_k b_{((i-k) \bmod n - j) \bmod n} \\ &= \sum_{k=0}^{n-1} a_k b_{((i-j) \bmod n - k) \bmod n} \\ &= (\boldsymbol{a} \circledast \boldsymbol{b})_{(i-j) \bmod n}. \end{aligned}$$

This suffices to show that the product of $C(\boldsymbol{a})$ and $C(\boldsymbol{b})$ yields a circulant matrix $C(\boldsymbol{z})$, with $\boldsymbol{z} = \boldsymbol{a} \circledast \boldsymbol{b}$. If circulant matrices are represented by storing only their first column, then the representation of their product can be computed in $O(n \log n)$ time using the FFT algorithm.

(b) Consider the system

$$C(\boldsymbol{a}) \times \boldsymbol{x} = \boldsymbol{b}, \tag{3.15}$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are arbitrary complex vectors and $\boldsymbol{x}$ is the vector of the unknowns. Observe that

$$\begin{aligned} [C(\boldsymbol{a}) \times \boldsymbol{x}]_i &= \sum_{k=0}^{n-1} a_{(i-k) \bmod n} x_k \\ &= (\boldsymbol{a} \circledast \boldsymbol{x})_i. \end{aligned}$$

Let F_n be the Fourier matrix of order n, and let $\boldsymbol{A}$, $\boldsymbol{X}$, $\boldsymbol{B}$ denote, respectively, $F_n\boldsymbol{a}$, $F_n\boldsymbol{x}$ and $F_n\boldsymbol{b}$. By the cyclic convolution theorem, System 3.15 is equivalent to the following system

$$\boldsymbol{A} \odot \boldsymbol{X} = \boldsymbol{B},$$

where $\odot$ denotes component-wise product. Note that such system consists of n equations, one for each component of the (unknown) vector $\boldsymbol{X}$. For $0 \leq i \leq n-1$, the i-th equation is

$$A_i X_i = B_i, \tag{3.16}$$

hence it contains the single unknown X_i. Therefore it immediately follows that

1. The system has one and only solution iff $A_i \neq 0$, for $0 \leq i \leq n-1$.
2. The system has no solution iff there exists an index i such that $A_i = 0$ and $B_i \neq 0$.
3. The system has infinite solutions iff for each index i such that $A_i = 0$, then $B_i = 0$, and at least one such index exists.

By the equivalence of Systems 3.16 and 3.15, the above considerations also apply to our original system. Note that $\boldsymbol{A}$ and $\boldsymbol{B}$ can be computed in $O(n \log n)$ time using the FFT algorithm, and that the subsequent test (as specified in Points 1..3 above) can be performed in additional $O(n)$ time.

Consider the case when at least one solution exists and define $\overline{\boldsymbol{X}}$ as

$$\overline{X}_i = \begin{cases} B_i/A_i & \text{if } A_i \neq 0 \\ 0 & \text{otherwise,} \end{cases}$$

for $0 \leq i \leq n-1$. Then, a solution to System 3.15 can be computed in $O(n \log n)$ time as $\overline{\boldsymbol{x}} = F_n^{-1}\overline{\boldsymbol{X}}$.

(c) By a well known theorem in linear algebra, $C(\boldsymbol{a})$ is invertible if and only if the linear system

$$C(\boldsymbol{a})\boldsymbol{x} = \boldsymbol{b}$$

has one and only solution $\boldsymbol{x} \in C^n$, for any given vector $\boldsymbol{b} \in C^n$. By the results in Part (b), we can therefore conclude that $C(\boldsymbol{a})$ is invertible if and only if $A_i = (F_n\boldsymbol{a})_i \neq 0$ for $0 \leq i \leq n-1$. Such condition can clearly be tested in $O(n \log n)$ time using the FFT algorithm to compute $\boldsymbol{A}$ and a linear scan to test that $A_i \neq 0$, for $0 \leq i \leq n-1$.

Let us now prove that $[C(\boldsymbol{a})]^{-1}$ is itself a circulant matrix. To see this, consider the system

$$[C(\boldsymbol{a})]\boldsymbol{x} = (1, 0, \ldots, 0). \tag{3.17}$$

Since $C(\boldsymbol{a})$ is invertible, System (3.17) has one and only solution $\overline{\boldsymbol{x}}$ such that

$$[C(\boldsymbol{a})]\overline{\boldsymbol{x}} = \overline{\boldsymbol{x}} \circledast \boldsymbol{a} = (1, 0, \ldots, 0).$$

Consider now the circulant matrix $C(\overline{\boldsymbol{x}})$. We have

$$C(\boldsymbol{a}) \times C(\overline{\boldsymbol{x}}) = C\left(\boldsymbol{a} \circledast \overline{\boldsymbol{x}}\right) = C(\ (1, 0, \ldots, 0)\ ) = I_n,$$

where I_n is the $n \times n$ identity matrix. Therefore

$$[C(\boldsymbol{a})]^{-1} = C(\overline{\boldsymbol{x}})$$

by the uniqueness of the inverse matrix. As shown in Part (b), System (3.17) can be solved in time $O(n \log n)$, therefore the inverse of $C(\boldsymbol{a})$, can be computed within the same time.

3.5 Transforms of (n, k)-sparse vectors

Let n and k be powers of two, with $1 \leq k \leq n$. We say that a vector $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$ is (n, k)-*sparse* if $x_i = 0$ for $i \bmod k \neq 0$, with $0 \leq i \leq n-1$. In this section, we design and analyze an algorithm which, on input an (n, k)-sparse vector $\boldsymbol{x}$, returns $DFT_n(\boldsymbol{x})$ in $O\left(n + \frac{n}{k} \log \frac{n}{k}\right)$ time, where the cost model assigns unit time to arithmetic operations and assignments between complex scalars.

We follow a approach different from the one seen in class (which used the Cooley-Tukey algorithm to determine the structure of the transform of an (n, k)-sparse vector) and, rather, modify the recursive FFT algorithm to suit the special structure of an (n, k)-sparse vector. Let $1 < k \leq n$, and let $\boldsymbol{x}^{[0]} = (x_0, x_2, \ldots, x_{n-2})$ and $\boldsymbol{x}^{[1]} = (x_1, x_3, \ldots, x_{n-1})$ be the two $n/2$-vectors containing, respectively, the even-indexed and odd-indexed components of $\boldsymbol{x}$. The immediate observation upon which we can base our algorithm is the following: if $\boldsymbol{x}$ is (n, k)-sparse, then $\boldsymbol{x}^{[0]}$ is $(n/2, k/2)$-sparse, while $\boldsymbol{x}^{[1]}$ is the null vector. To see this, observe that, for $0 \leq i \leq n/2 - 1$, $x_i^{[0]} = 0$ for $2i \bmod k \neq 0$, and the latter implies that $i \bmod (k/2) \neq 0$, while $x_i^{[1]} = x_{2i+1} = 0$, since no odd index can be a multiple of $k > 1$, a power of two. Recall that the FFT algorithm computes $DFT_n(\boldsymbol{x})$ by recursively computing $DFT_{n/2}(\boldsymbol{x}^{[0]})$ and $DFT_{n/2}(\boldsymbol{x}^{[1]})$ and then performing another $\Theta(n)$ additional operations. When invoked on an (n, k)-sparse vector with $k > 1$, the algorithm can save one recursive call, since only $DFT_{n/2}(\boldsymbol{x}^{[0]})$ must be computed. Moreover the recombination work becomes straightforward. When $k = 1$, there is no special structure of the vector and the standard FFT algorithm is executed instead. The algorithm follows.

```
SPARSE_FFT(x, k)
n ← length(x)
if k = 1 then return FFT(x)
x^[0] ← (x_0, x_2, ..., x_{n-2})
y^[0] ← SPARSE_FFT(x^[0], k/2)
for i ← 0 to n/2 − 1 do
    y_i ← y_i^[0]
    y_{i+n/2} ← y_i^[0]
return y
```

Note that at each recursive step, $\boldsymbol{y}$ is obtained as the concatenation of $\boldsymbol{y}^{[0]}$ with itself. It follows that the DFT_n of an (n, k)-sparse vector is the k-fold repetition of the $DFT_{n/k}$ of the vector containing its n/k components of index ki, with $i = 0, 1, n/k - 1$.

The correctness of the above algorithm follows from the correctness of the FFT algorithm and the observations made above. As for its running time, we can write the following

recurrence in n and k:

$$\begin{aligned} T(n,k) &= T\left(\frac{n}{2},\frac{k}{2}\right)+n,\ n\geq k>1,\\ T(n,1) &= \Theta(n\log n). \end{aligned}$$

By iterating the above recurrence we obtain:

$$\begin{aligned} T(n,k) &= T\left(\frac{n}{2},\frac{k}{2}\right)+n\\ &= T\left(\frac{n}{4},\frac{k}{4}\right)+n+\frac{n}{2}\\ &\vdots\\ &= T\left(\frac{n}{k},1\right)+n\sum_{i=0}^{\log k-1}2^{-i}\\ &= O\left(\frac{n}{k}\log\frac{n}{k}+n\right). \end{aligned}$$

As a last observation, observe that the work in the above recurrence is completely due to assignments, while aroithmetic operations account for the work at the (single) leave. As a consequence, if the cost model did not charge assignments, the running time (i.e., the number of scalar complex arithmetic operations) would be $\Theta((n/k/\log(n/k))$.

Exercise 3.2 The complex number $\omega = e^{i\pi/4} = \frac{\sqrt{2}}{2}+\frac{i\sqrt{2}}{2}$ is an 8-th principal root of unity in the complex field.

(a) Compute ω^i for $i = 0,1,2,3,4,5,6,7$.

(b) Write the Fourier transform matrix $F_8 = [\omega^{ij}]$ for $i,j = 0,1,\ldots,7$.

(c) Write $F_8^{-1} = (1/8)[\omega^{-ij}]$ for $i,j = 0,1,\cdots,7$.

(d) Compute $\boldsymbol{X} = F_8\boldsymbol{x}$, for $\boldsymbol{x} = (1,0,1,-1,0,0,-1,1)$.

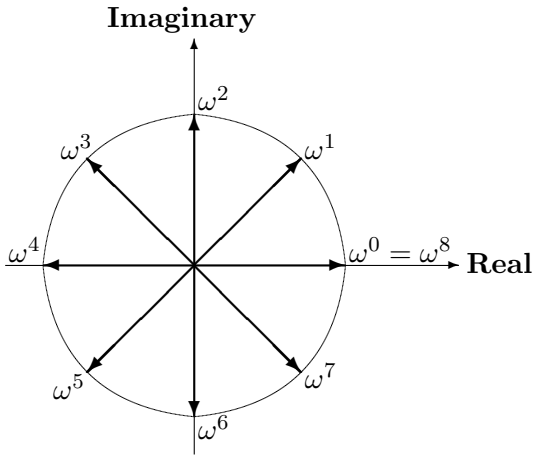
(e) Let $\boldsymbol{x} = (0,0,1,0,0,0,0,0)$ and $\boldsymbol{y} = (0,0,0,0,1,0,0,0)$. Compute the cyclic convolution $\boldsymbol{z} = \boldsymbol{x}\circledast\boldsymbol{y}$ using the definition.

(f) Repeat (e) using the cyclic convolution theorem.

Answer:

(a) Let $d = \frac{\sqrt{2}}{2}$. We have:

$$\begin{aligned}
&\omega^0 = 1 \qquad\qquad \omega^1 = (d + di)\\
&\omega^2 = (d + di)^2 = d^2 + 2\tfrac{1}{2}i - d^2 = i\\
&\omega^3 = \omega^2(d + di) = di + di^2 = -d + di\\
&\omega^4 = \omega^3(d + di) = -d^2 + (di)^2 = -1\\
&\omega^5 = \omega^4(d + di) = -d - di\\
&\omega^6 = \omega^5(d + di) = -d^2 - 2d^2i - \\
&\qquad\qquad\qquad\qquad -(di)^2 = -i\\
&\omega^7 = \omega^6(d + di) = -di - di^2 = d - di
\end{aligned}$$



For higher powers ω^r with $r > 7$, recall that $\omega^r = \omega^{r \bmod 8}$.

(b)

$$F_8 = \begin{bmatrix}
\omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0\\
\omega^0 & \omega^1 & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7\\
\omega^0 & \omega^2 & \omega^4 & \omega^6 & \omega^8 & \omega^{10} & \omega^{12} & \omega^{14}\\
\omega^0 & \omega^3 & \omega^6 & \omega^9 & \omega^{12} & \omega^{15} & \omega^{18} & \omega^{21}\\
\omega^0 & \omega^4 & \omega^8 & \omega^{12} & \omega^{16} & \omega^{20} & \omega^{24} & \omega^{28}\\
\omega^0 & \omega^5 & \omega^{10} & \omega^{15} & \omega^{20} & \omega^{25} & \omega^{30} & \omega^{35}\\
\omega^0 & \omega^6 & \omega^{12} & \omega^{18} & \omega^{24} & \omega^{30} & \omega^{36} & \omega^{42}\\
\omega^0 & \omega^7 & \omega^{14} & \omega^{21} & \omega^{28} & \omega^{35} & \omega^{42} & \omega^{49}
\end{bmatrix}$$

$$= \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
1 & (d + di) & i & (-d + di) & -1 & (-d - di) & -i & (d - di)\\
1 & i & -1 & -i & 1 & i & -1 & -i\\
1 & (-d + di) & -i & (d + di) & -1 & (d - di) & i & (-d - di)\\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1\\
1 & -(d - di) & i & (d - di) & -1 & (d + di) & -i & (-d + di)\\
1 & -i & -1 & i & 1 & -i & -1 & i\\
1 & (d - di) & -i & (-d - di) & -1 & (-d + di) & i & (d + di)
\end{bmatrix}$$

(c)

$$
\begin{aligned}
F_8^{-1} &= \frac{1}{8}\begin{bmatrix}
\omega^{-0} & \omega^{-0} & \omega^{-0} & \omega^{-0} & \omega^{-0} & \omega^{-0} & \omega^{-0} & \omega^{-0} \\
\omega^{-0} & \omega^{-1} & \omega^{-2} & \omega^{-3} & \omega^{-4} & \omega^{-5} & \omega^{-6} & \omega^{-7} \\
\omega^{-0} & \omega^{-2} & \omega^{-4} & \omega^{-6} & \omega^{-8} & \omega^{-10} & \omega^{-12} & \omega^{-14} \\
\omega^{-0} & \omega^{-3} & \omega^{-6} & \omega^{-9} & \omega^{-12} & \omega^{-15} & \omega^{-18} & \omega^{-21} \\
\omega^{-0} & \omega^{-4} & \omega^{-8} & \omega^{-12} & \omega^{-16} & \omega^{-20} & \omega^{-24} & \omega^{-28} \\
\omega^{-0} & \omega^{-5} & \omega^{-10} & \omega^{-15} & \omega^{-20} & \omega^{-25} & \omega^{-30} & \omega^{-35} \\
\omega^{-0} & \omega^{-6} & \omega^{-12} & \omega^{-18} & \omega^{-24} & \omega^{-30} & \omega^{-36} & \omega^{-42} \\
\omega^{-0} & \omega^{-7} & \omega^{-14} & \omega^{-21} & \omega^{-28} & \omega^{-35} & \omega^{-42} & \omega^{-49}
\end{bmatrix} \\
&= \frac{1}{8}\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & (d-di) & -i & (-d-di) & -1 & (-d+di) & i & (d+di) \\
1 & -i & -1 & i & 1 & -i & -1 & i \\
1 & (-d-di) & i & (d-di) & -1 & (d+di) & -i & (-d+di) \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1 & -(d-di) & -i & (d+di) & -1 & (d-di) & i & (-d-di) \\
1 & i & -1 & -i & 1 & i & -1 & -i \\
1 & (d+di) & i & (-d+di) & -1 & (-d-di) & -i & (d-di)
\end{bmatrix}
\end{aligned}
$$

(d)

$$
\begin{aligned}
\boldsymbol{X} &= F_8 \cdot \boldsymbol{x} = F_8 \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \\ -1 \\ 0 \\ 0 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix}
1+0+1-1+0+0-1+1 \\
1+0+i-(-d+di)+0+0-(-i)+(d-di) \\
1+0+(-1)-(-i)+0+0-(-1)+(-i) \\
1+0+(-i)-(d+di)+0+0-i+(-d-di) \\
1+0+1-(-1)+0+0-1+(-1) \\
1+0+i-(d-di)+0+0-(-i)+(-d+di) \\
1+0+(-1)-i+0+0-(-1)+i \\
1+0+(-i)-(-d-di)+0+0-i+(d+di)
\end{bmatrix} \\
&= \begin{bmatrix}
1 \\
(1+2d)+(2-2d)i \\
1 \\
(1-2d)+(-2-2d)i \\
1 \\
(1-2d)+(2+2d)i \\
1 \\
(1+2d)+(-2+2d)i
\end{bmatrix}
\end{aligned}
$$

(e) Let $n = 8$. Since only x_2 and y_4 are nonzero, the only way $x_l y_{(i-l) \bmod n}$ can be nonzero is when $l = 2$ and $i = 6$ (we get the value of l directly; for i, from $(i - l) \bmod n = 4$, we get $(i - 2) \bmod n = 4$, which implies $i = 6$). Thus, $z_6 = x_2 y_4 = 1$, while all the other z_i's are zero.

(f) First, we note that, since $\boldsymbol{x}$ and $\boldsymbol{y}$ have only one nonzero element each, and that the element is a 1, $F_8\boldsymbol{x}$ is column 2 of F_8 and $F_8\boldsymbol{y}$ is column 4 of F_8. Now, from the definition of F_8, we know that $[F_8]_{ij} = \omega^{i \cdot j}$; thus, the i-th element of $F_8\boldsymbol{x} \cdot F_8\boldsymbol{y}$ is $\omega^{i \cdot 2 + i \cdot 4} = \omega^{i \cdot 6}$. But this is exactly column 6 of F_8! Therefore, multiplying F_8^{-1} by $F_8\boldsymbol{x} \cdot F_8\boldsymbol{y}$ we obtain the column 6 of the identity matrix, namely $(0\,0\,0\,0\,0\,0\,1\,0)$, exactly the answer in Part (e). □

Exercise 3.3 Consider the linear convolution $\boldsymbol{u} \star \boldsymbol{x}$, where both sequences have length n and $\boldsymbol{u} = (1, 1, \ldots, 1)$. Design an algorithm that performs the above operation in time $O(n)$.

Answer: Let $\boldsymbol{w} = \boldsymbol{u} \star \boldsymbol{x}$. Recall that $\boldsymbol{w}$ has $2n - 1$ components and that

$$\begin{aligned} w_i &= \sum_{j=\max\{0,i-n+1\}}^{\min\{n-1,i\}} u_j x_{i-j} \\ &= \begin{cases} \sum_{j=0}^{i} x_j & \text{if } 0 \le i \le n-1, \\ \\ \sum_{j=i-n+1}^{n-1} x_j & \text{if } n \le i \le 2n-2. \end{cases} \end{aligned} \tag{3.18}$$

From (3.18) we can easily derive the following two recurrences.

$$\begin{cases} w_0 = x_0 \\ w_i = w_{i-1} + x_i, \quad 1 \le i \le n-1. \end{cases} \qquad \begin{cases} w_{2n-2} = x_{n-1} \\ w_{2n-2-i} = w_{2n-2-i+1} + x_{n-1-i}, \quad n-1 \ge i \ge 1. \end{cases}$$

(Note that both recurrences compute w_{n-1}). The algorithm is the following:

```
UNIT_LIN_CONV(x)
    n ← length(x)
    z_0 ← x_0
    z_{2n-2} ← x_{n-1}
    for i ← 1 to n − 1 do
        z_i ← z_{i−1} + x_i
        z_{2n-2-i} ← z_{2n-2-i+1} + x_{n-1-i}
    return z
```

The program performs exactly $2n - 2$ additions and therefore runs in linear time. □

Exercise 3.4 Let $n = pq$. Let $x = (x_0, x_1, \ldots, x_{n-1})$ be a *periodic sequence* of period q, that is, $x_{i+q} = x_i$, for $i = 0, 1, \ldots, n-1-q$. Let $X = (X_0, X_1, \ldots, X_{n-1}) = Fx$. Prove that $X_k = 0$ unless k is a multiple of p. (*Hint:* base your argument on the Cooley-Tukey algorithm introduced in Section 3.1).

Exercise 3.5 Consider the following equation system

$$\boldsymbol{x} \circledast \boldsymbol{x} = \boldsymbol{b},$$

where $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$ is a vector of complex unknowns and $\boldsymbol{b} = (b_0, b_1, \ldots, b_{n-1})$ is a given vector of complex numbers.

(a) How many solutions has the system? In case the number of solutions is a function of $\boldsymbol{b}$, derive this function.

(b) Give an $O(n \log n)$ algorithm that, on input $\boldsymbol{b}$, outputs *one* solution to the system, if one exists.

Exercise 3.6 The complex number $\omega_3 = -1/2 + i\sqrt{3}/2$ is the principal third root of the unity in the complex field.

(a) Evaluate ω_3^i for $i = 0, 1, 2, 3, 4$.

(b) Write the Fourier matrix F_3.

(c) Write F_3^{-1}

(d) Let $\boldsymbol{x} = (0, 1, 2)$. Let $\boldsymbol{y}$ be the cyclic convolution of 8 vectors all equal to $\boldsymbol{x}$. Compute $\boldsymbol{y}$.

Exercise 3.7 Let $(X_0, X_1, \ldots, X_{n-1}) = DFT_n(x_0, x_1, \ldots, x_{n-1})$. Consider now the vector $(Y_0, Y_1, \ldots, Y_{2n-1}) = DFT_{2n}(x_0, 0, x_1, 0, \ldots, x_{n-1}, 0)$. Write the Y_k's as a function of the X_i's.

Exercise 3.8 Let m and n be two integers, with $m \gg n$ a multiple of n. Describe and analyze an algorithm to multiply the two polynomials $p(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_0$ and $q(x) = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \cdots + b_0$ in time $O(m \log n)$.