5. THE FAST FOURIER TRANSFORM

Raveen de Silva, r.desilva@unsw.edu.au
office: K17 202
Course Admin: Anahita Namvar, cs3121@cse.unsw.edu.au

School of Computer Science and Engineering
UNSW Sydney

Term 3, 2021

Table of Contents

Our strategy to multiply polynomials fast

- Given two polynomials of degree at most n,

$$P_A(x) = A_n x^n + \ldots + A_0 \text{ and } P_B(x) = B_n x^n + \ldots + B_0,$$

we want to find $P_C(x) = P_A(x)\, P_B(x)$.

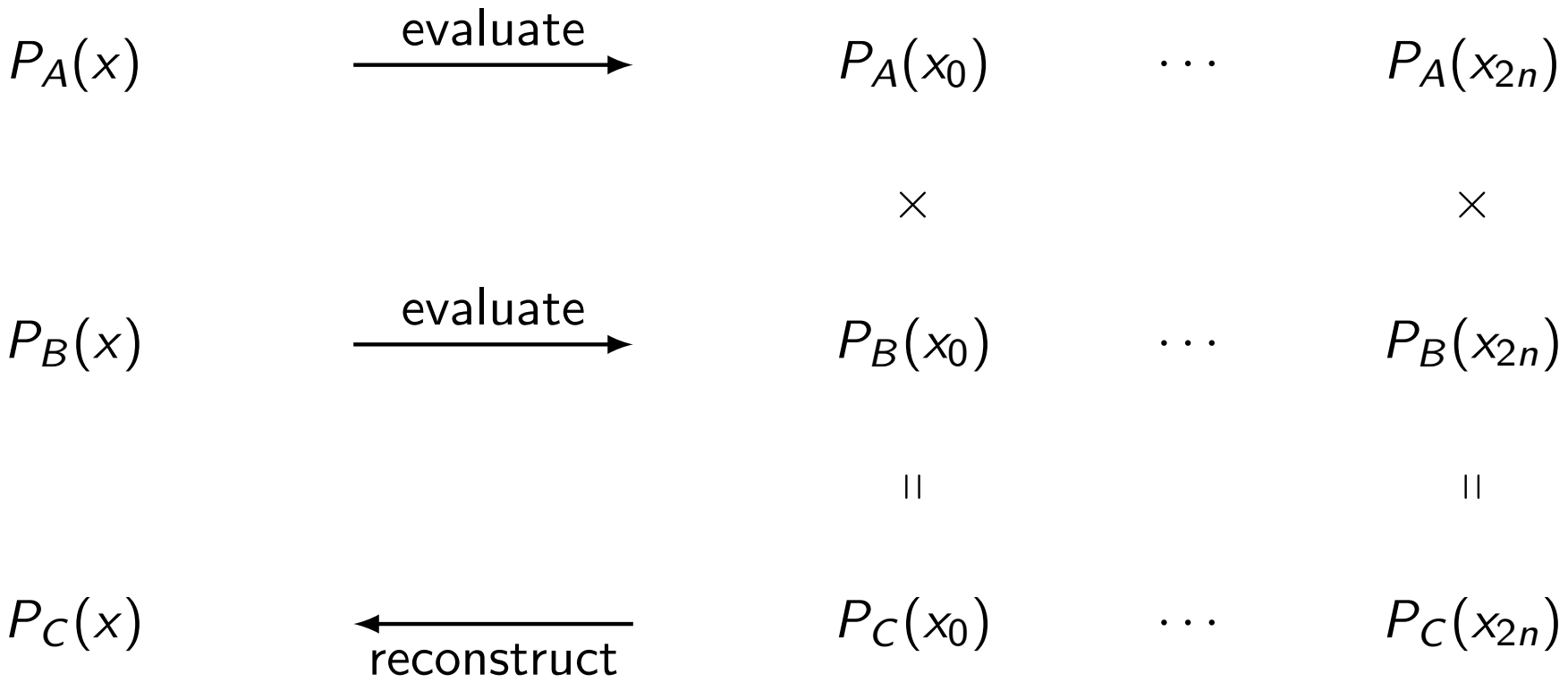
1. Evaluate $P_A(x)$ and $P_B(x)$ at $2n+1$ distinct points

$$x_0, x_1, \ldots, x_{2n}$$

2. Multiply them point by point using $2n+1$ multiplications of large numbers, to get $2n+1$ values of $P_C(x)$
3. Reconstruct the coefficients of $P_C(x)$, i.e.

$$P_C(x) = C_{2n} x^{2n} + C_{2n-1} x^{2n-1} + \ldots + C_1 x + C_0.$$

Our strategy to multiply polynomials fast



Our strategy to multiply polynomials fast

- Previously, we chose $x_0, \ldots, x_{2n}$ to be the $2n+1$ integers which are the smallest by their absolute value, i.e.,

$$-n, -(n-1), \ldots, -1, 0, 1, \ldots, n-1, n.$$

- However, this choice requires us to compute values such as

$$P_A(n) = A_0 + A_1 n + \ldots + A_{n-1} n^{n-1} + A_n n^n$$

- As the degree n of the polynomials $P_A(x)$ and $P_B(x)$ increases, the value of n^n increases very fast and causes rapid increase of the computational complexity of the algorithm for polynomial multiplication which we used in the generalised Karatsuba algorithm.

Our strategy to multiply polynomials fast

Question

What values should we take for $x_0, \ldots, x_{2n}$ to avoid "explosion" of size when we evaluate x_i^n while computing $P_A(x_i) = A_0 + A_1 x + \ldots + A_n x_i^n$?

Answer

We would like $|x_i^n| = 1$, but this can't be achieved with only real numbers ...

Complex numbers revisited

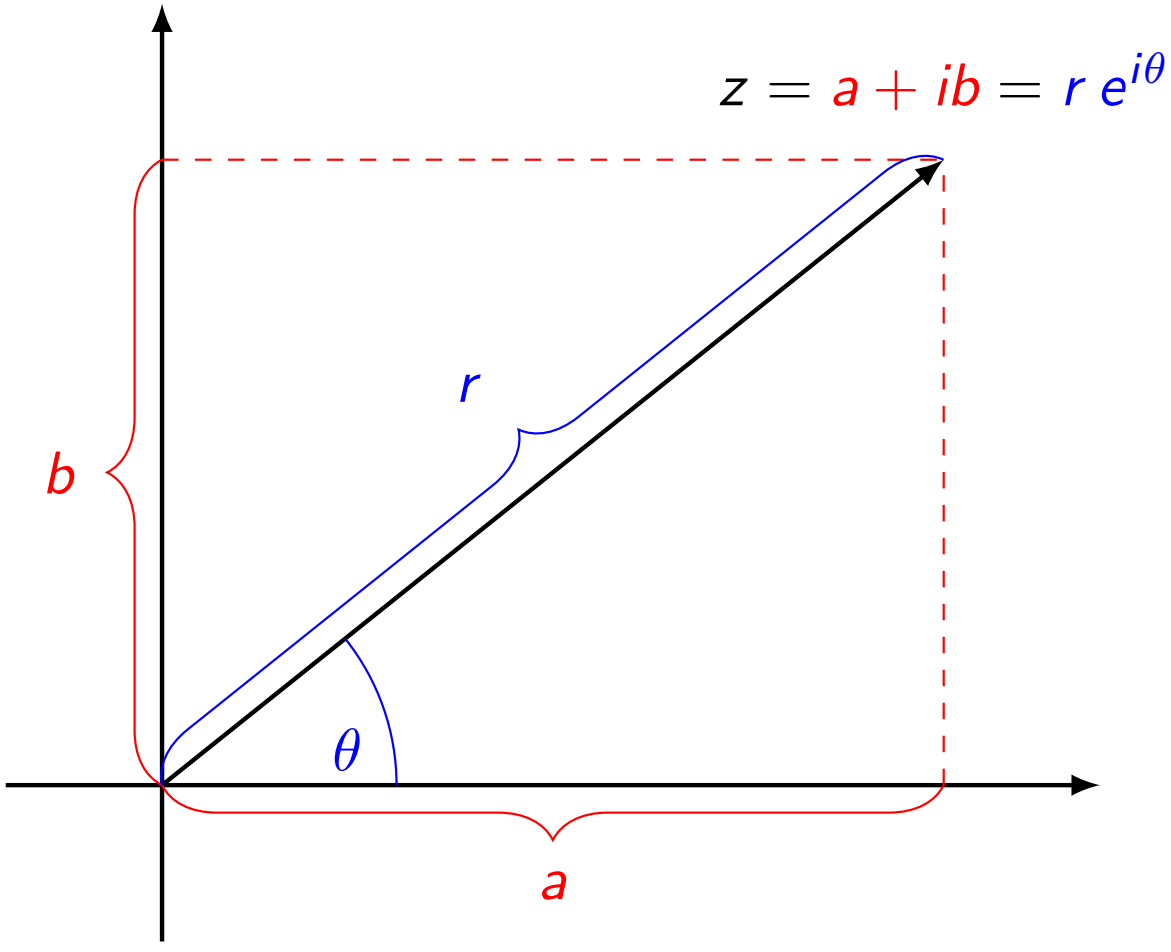
Let $z = a + ib$ be a complex number, where $a, b \in \mathbb{R}$. Then we can define the

- *modulus* $|z|$, a positive real number r such that $r = \sqrt{a^2 + b^2}$ and
- *argument* $\arg z$, an angle $\theta \in (-\pi, \pi]$ such that $a = r\cos\theta$ and $b = r\sin\theta$.

Then we have

$$\begin{aligned} z &= a + ib \\ &= r(\cos\theta + i\sin\theta) \\ &= r\,e^{i\theta}. \end{aligned}$$

Complex numbers revisited



Complex numbers revisited

To multiply two complex numbers, we multiply the moduli and add the arguments.

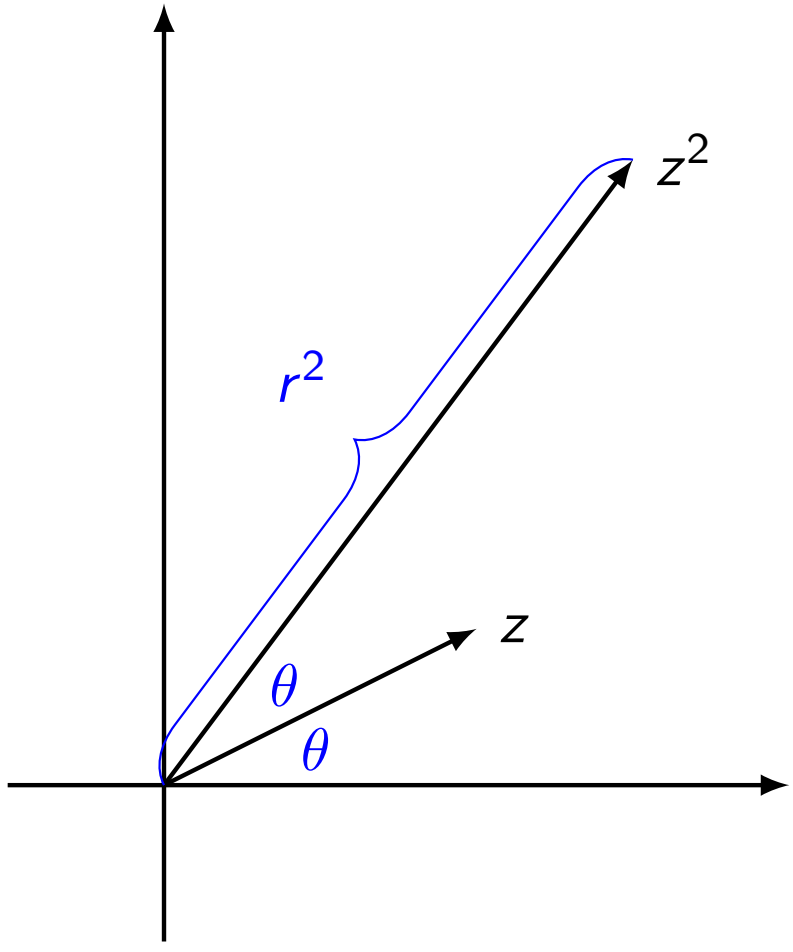
Theorem

If $z = r\,e^{i\theta}$ and $w = s\,e^{i\phi}$, then $z\,w = (r\,s)\,e^{i\,(\theta+\phi)}$.

Corollary

In particular, if $z = r\,e^{i\theta}$ then $z^n = r^n\,e^{i\,(n\theta)}$.

Complex numbers revisited



Complex roots of unity

- We are interested in the solutions of $z^m = 1$, known as *roots of unity of order m*.
- Solving $r^m\, e^{i\,(m\theta)} = 1$, we see that

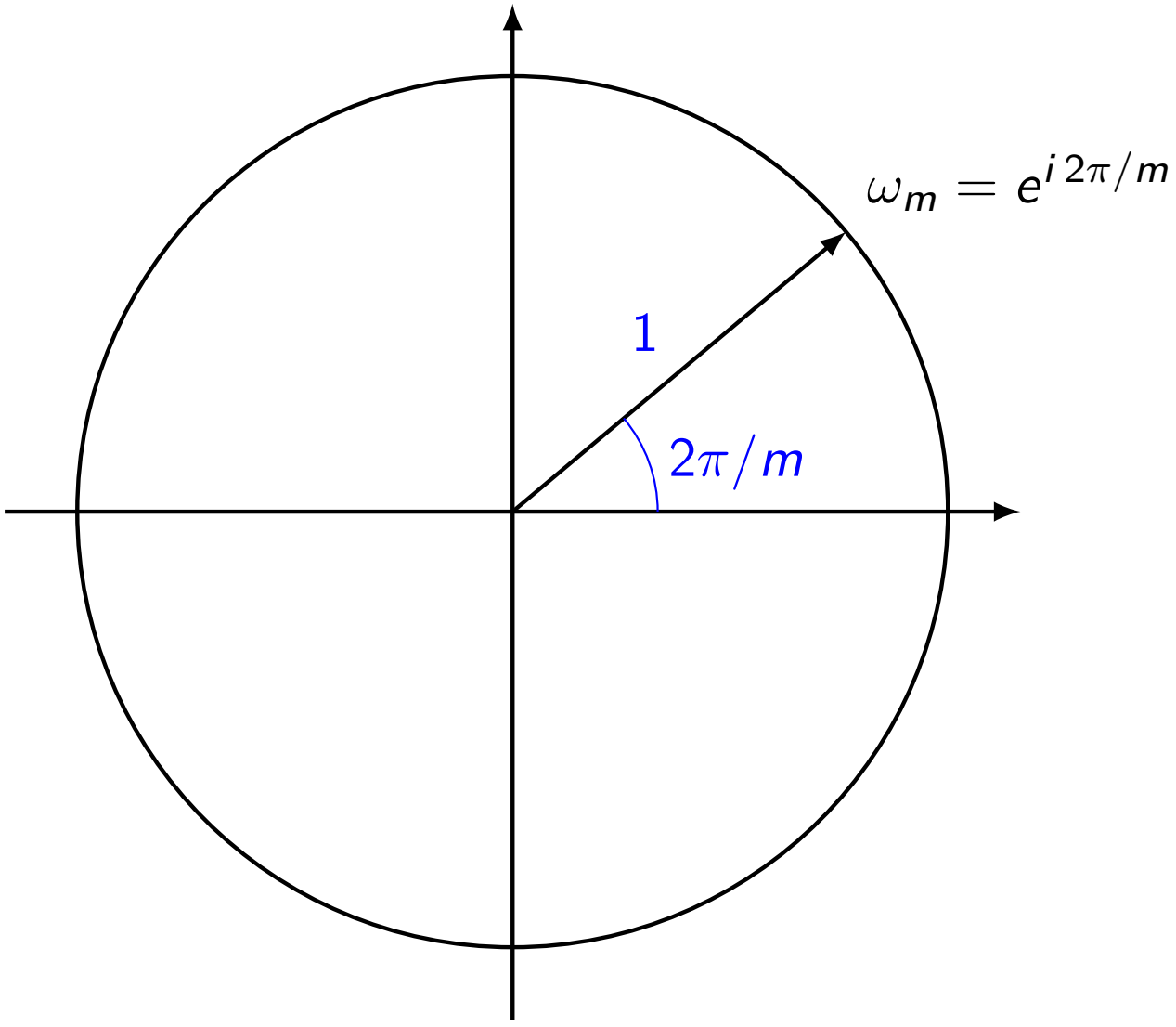
$$r^m = 1, \text{ i.e. } r = 1$$

and

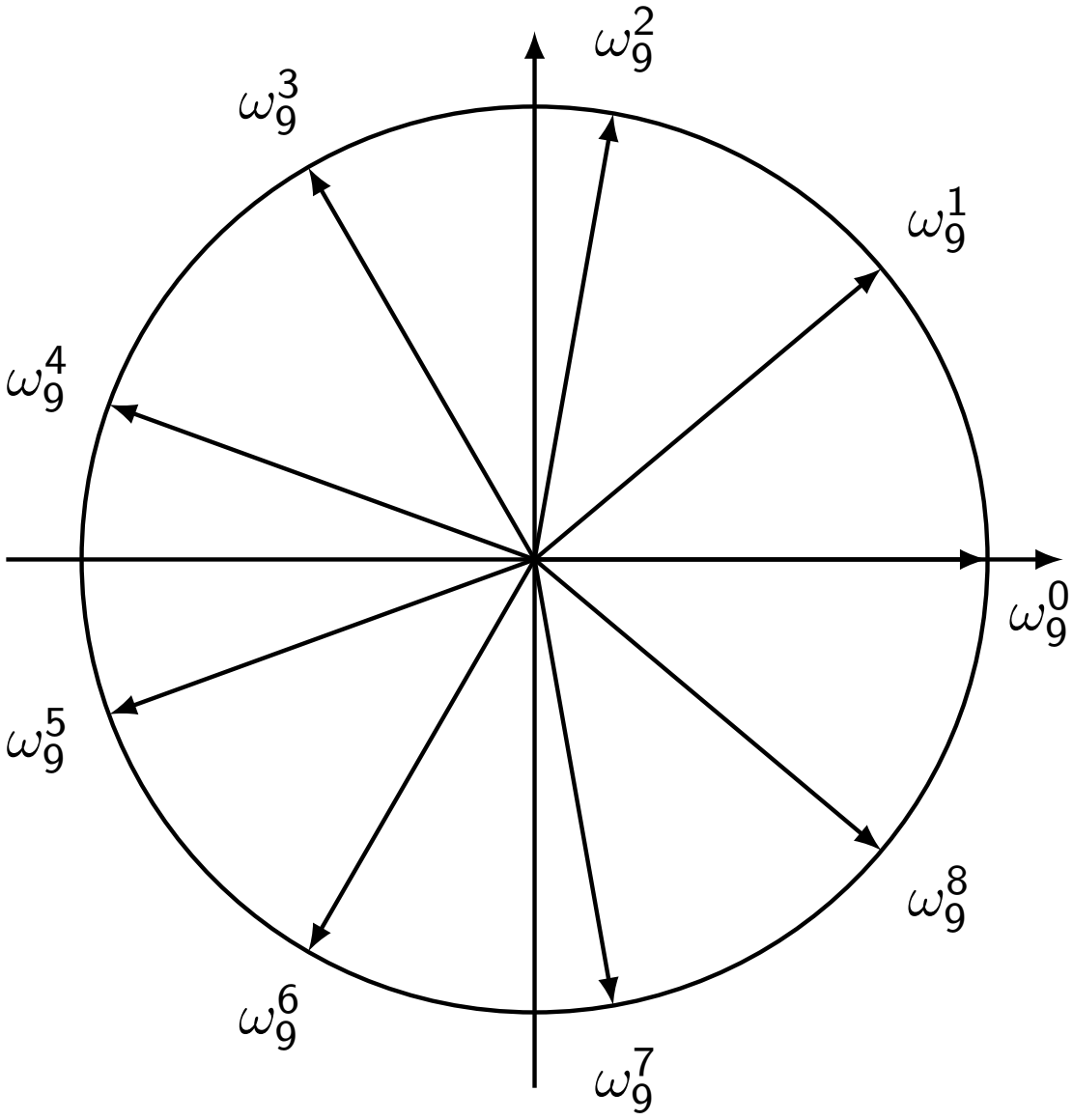
$$m\theta = 2\pi\, k \text{ i.e. } \theta = \frac{2\pi\, k}{m}.$$

- Let $\omega_m = e^{i\,2\pi/m}$. Then $z = \omega_m^k$, i.e., all roots of unity of order m can be written as powers of ω_m. We say that ω_m is a *primitive root of unity of order m*.
- Note that there are only m distinct values here, as $\omega_m^m = \omega_m^0$.

Complex roots of unity



Complex roots of unity



Complex roots of unity

- For $\omega_m = e^{i\,2\pi/m}$ and for all k such that $0 \leq k \leq m-1$,

$$((\omega_m)^k)^m = (\omega_m)^{m\,k} = ((\omega_m)^m)^k = 1^k = 1.$$

- Thus, $\omega_m^k = (\omega_m)^k$ is also a root of unity.

Theorem

ω_m^k is a primitive root of unity of order n if and only if $\gcd(m, k) = 1$.

Complex roots of unity

- The product of two roots of unity of order m is given by

$$\omega_m^i \omega_m^j = \omega_m^{i+j},$$

  which is itself a root of unity of the same order.
- So the set of all roots of unity of order m, i.e., $\{1, \omega_m, \omega_m^2, \ldots, \omega_m^{m-1}\}$ is closed under multiplication (and by extension, under taking powers).
- Note that this is not true for addition, i.e., the sum of two roots of unity is NOT another root of unity!

Note

This is an example of a *group*.

Complex roots of unity

Cancellation Lemma

For all positive integers k, m and integers ℓ, $\omega_{km}^{k\ell} = \omega_m^{\ell}$.

Proof

$$\omega_{km}^{k\ell} = \left(e^{i\frac{2\pi}{km}}\right)^{k\ell} = e^{i\frac{2\pi k\ell}{km}} = e^{i\frac{2\pi\ell}{m}} = \left(e^{i\frac{2\pi}{m}}\right)^{\ell} = \omega_m^{\ell}.$$

- In particular, $(\omega_{2m}^k)^2 = \omega_{2m}^{2k} = (\omega_{2m}^2)^k = \omega_m^k$, i.e. the squares of the roots of unity of order $2m$ are just the roots of unity of order m.

Table of Contents

A new way to multiply polynomials fast

- Given two polynomials of degree at most n,

$$P_A(x) = A_n x^n + \ldots + A_0 \text{ and } P_B(x) = B_n x^n + \ldots + B_0,$$

we want to find $P_C(x) = P_A(x)\,P_B(x)$.

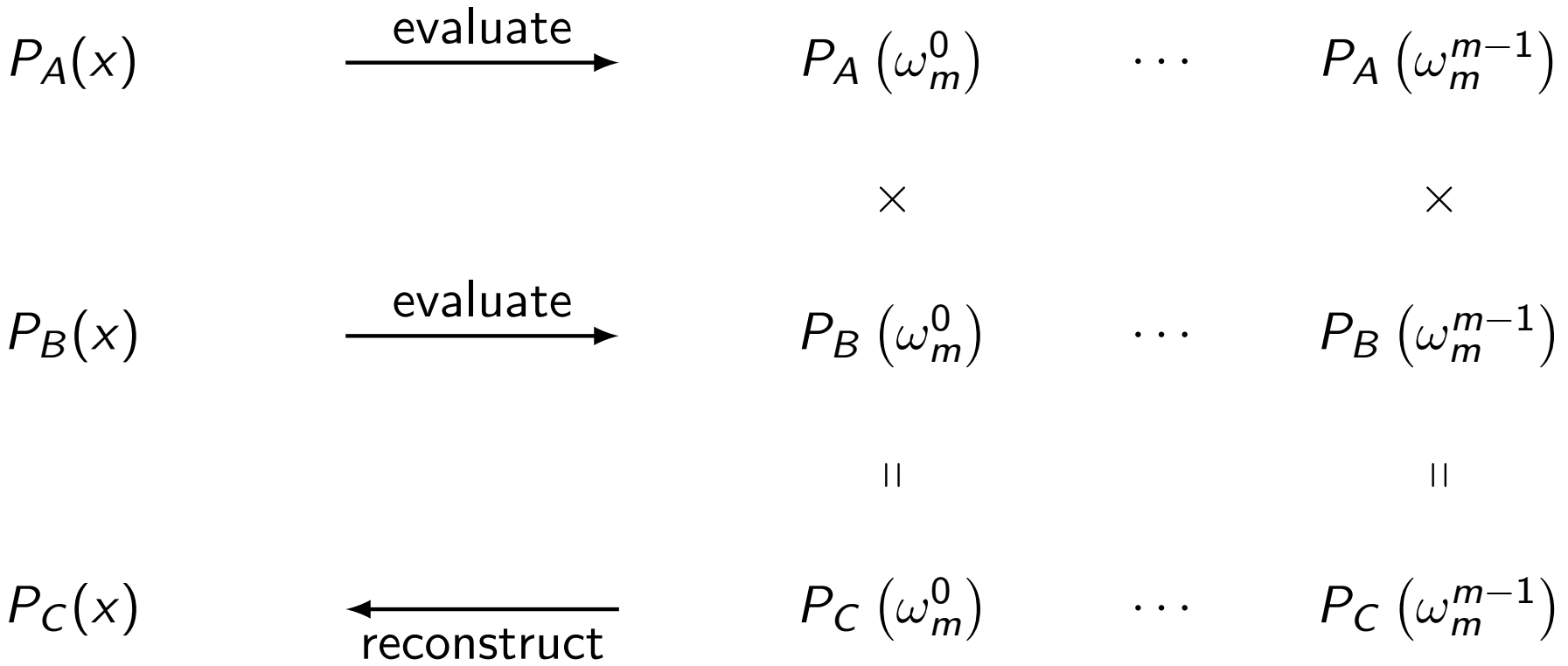
1. Evaluate $P_A(x)$ and $P_B(x)$ at $m = 2n+1$ distinct points

$$\omega_m^0, \omega_m^1, \ldots, \omega_m^{m-1}$$

2. Multiply them point by point using $2n+1$ multiplications of large numbers, to get $2n+1$ values of $P_C(x)$
3. Reconstruct the coefficients of $P_C(x)$, i.e.

$$P_C(x) = C_{2n} x^{2n} + C_{2n-1} x^{2n-1} + \ldots + C_1 x + C_0.$$

Our strategy to multiply polynomials fast



The Discrete Fourier Transform

Definition

Let $A = \langle A_0, A_1, \ldots, A_{m-1} \rangle$ be a sequence of m real or complex numbers, and let the corresponding polynomial be $P_A(x) = \sum_{j=0}^{m-1} A_j x^j$.

Let $\widehat{A}_k = P_A(\omega_m^k)$ for all $0 \leq k \leq m-1$, the values of P_A at the roots of unity of order m.

The sequence of values $\widehat{A} = \langle \widehat{A}_0, \widehat{A}_1, \ldots, \widehat{A}_{m-1} \rangle$ is called the **Discrete Fourier Transform (DFT)** of A.

The Discrete Fourier Transform

- Our polynomials $P_A(x)$ and $P_B(x)$ have degree n, so the corresponding sequences of coefficients have only $n+1$ terms.
- However, we defined the DFT of a sequence as having the same length as the original sequence, and we must obtain the values at all m roots of unity of order m.
- This is easily rectified by padding the sequences A and B with zeros at the end, corresponding to the terms $0x^{n+1} + 0x^{n+2} + \ldots + 0x^{2n}$, since $m = 2n+1$.
- The DFT $\widehat{A}$ of a sequence A can be computed VERY FAST using a divide-and-conquer algorithm called the **Fast Fourier Transform**.

Our strategy to multiply polynomials fast

- We can now compute the DFTs of the two (0 padded) sequences:

$$DFT(\langle A_0, A_1, \ldots, A_n, \underbrace{0, \ldots, 0}_{n}\rangle) = \langle \widehat{A}_0, \widehat{A}_1, \ldots, \widehat{A}_{m-1}\rangle$$

  and

$$DFT(\langle B_0, B_1, \ldots, B_n, \underbrace{0, \ldots, 0}_{n}\rangle) = \langle \widehat{B}_0, \widehat{B}_1, \ldots, \widehat{B}_{m-1}\rangle$$

- For each k we multiply the corresponding values $\widehat{A}_k = P_A(\omega_m^k)$ and $\widehat{B}_k = P_B(\omega_m^k)$, thus obtaining

$$\widehat{C}_k = \widehat{A}_k \widehat{B}_k = P_A(\omega_m^k) P_B(\omega_m^k) = P_C(\omega_m^k)$$

- We then use the inverse transformation for DFT, called IDFT, to recover the coefficients $\langle C_0, C_1, \ldots, C_{m-1}\rangle$ of the product polynomial $P_C(x)$ from the sequence $\langle \widehat{C}_0, \widehat{C}_1, \ldots, \widehat{C}_{m-1}\rangle$ of its

Our strategy to multiply polynomials fast

$$\langle A_0, A_1, \ldots, A_n, \underbrace{0, \ldots, 0}_{n} \rangle \xrightarrow{\text{DFT}} \widehat{A}_0 \quad \cdots \quad \widehat{A}_{m-1}$$

$$\times \qquad\qquad \times$$

$$\langle B_0, B_1, \ldots, B_n, \underbrace{0, \ldots, 0}_{n} \rangle \xrightarrow{\text{DFT}} \widehat{B}_0 \quad \cdots \quad \widehat{B}_{m-1}$$

$$= \qquad\qquad =$$

$$\langle C_0, C_1, \ldots, C_{m-1} \rangle \xleftarrow{\text{IDFT}} \widehat{C}_0 \quad \cdots \quad \widehat{C}_{m-1}$$

The Fast Fourier Transform (FFT)

Question

How long does it take to compute the DFT of a sequence A, i.e. find the values

$$P_A(\omega_m^k) = A_0 + A_1\,\omega_m^k + A_2\,(\omega_m^k)^2 + \ldots + A_{m-1}\,(\omega_m^k)^{m-1}$$

for all k such that $0 \leq k < m$?

Answer

Even if we were to precompute all powers of ω_m (recall, there are only m distinct values), there are m multiplications required for each of m values $P_A(\omega_m^k)$.

However, we can use divide-and-conquer to find all the required values in $O(n \log n)$ time!

The Fast Fourier Transform (FFT)

- We can assume that m is a power of 2 - otherwise we can pad $P_A(x)$ with zero coefficients until its number of coefficients becomes equal to the nearest power of 2.

Exercise

Show that for every m which is not a power of two the smallest power of 2 larger or equal to m is smaller than $2m$.

Hint

Consider m in binary. How many bits does the nearest power of two have?

The Fast Fourier Transform (FFT)

Problem

Given a sequence $A = \langle A_0, A_1, \ldots, A_{m-1} \rangle$, compute its DFT, i.e. find the values $P_A(x)$ at $x = \omega_m^k$ for all k such that $0 \leq k < m$.

The main idea

We use divide-and-conquer by splitting the polynomial $P_A(x)$ into the even powers and the odd powers:

$$\begin{aligned} P_A(x) &= (A_0 + A_2\,x^2 + A_4\,x^4 + \ldots + A_{m-2}\,x^{m-2}) \\ &\quad + (A_1\,x + A_3\,x^3 + \ldots + A_{m-1}\,x^{m-1}) \\ &= A_0 + A_2\,x^2 + A_4\,(x^2)^2 + \ldots + A_{m-2}\,(x^2)^{m/2-1} \\ &\quad + x\left(A_1 + A_3\,x^2 + A_5\,(x^2)^2 + \ldots + A_{m-1}\,(x^2)^{m/2-1}\right) \end{aligned}$$

The Fast Fourier Transform (FFT)

- Let us define $A^{[0]} = \langle A_0, A_2, A_4, \ldots A_{m-2} \rangle$ and $A^{[1]} = \langle A_1, A_3, A_5, \ldots A_{m-1} \rangle$, so that

$$P_{A^{[0]}}(y) = A_0 + A_2\, y + A_4\, y^2 + \ldots + A_{m-2}\, y^{m/2-1}$$
$$P_{A^{[1]}}(y) = A_1 + A_3\, y + A_5\, y^2 + \ldots + A_{m-1}\, y^{m/2-1}$$

  and hence

$$P_A(x) = P_{A^{[0]}}(x^2) + x\, P_{A^{[1]}}(x^2)$$

- Note that the polynomials $P_{A^{[0]}}(y)$ and $P_{A^{[1]}}(y)$ have $m/2$ coefficients each, while the polynomial $P_A(x)$ has m coefficients.

The Fast Fourier Transform (FFT)

Question

Have we successfully reduced a problem of size m to a problem of size $m/2$?

- **Problem of size m:**
  Evaluate a polynomial with m coefficients at all roots of unity of order m.

- **Problem of size $m/2$:**
  Evaluate a polynomial with $m/2$ coefficients at all roots of unity of order $m/2$.

The Fast Fourier Transform (FFT)

- The original problem was to evaluate the polynomial $P_A(x)$ with m coefficients at inputs $x = \omega_m^k$ for each $0 \leq k < m$. We have reduced it to the evaluation of two polynomials $P_{A^{[0]}}(y)$ and $P_{A^{[1]}}(y)$ each with $m/2$ coefficients at points

$$y = x^2 = \omega_m^{2k}.$$

- Recall that we assumed m is a power of 2, and hence even. We can therefore use the cancellation lemma

$$y = \omega_m^{2k} = \omega_{m/2}^k$$

to deduce that y is a root of unity of order $m/2$.

- Note that k goes up to $m-1$, but there are only $m/2$ distinct roots of unity of order $m/2$. Since $\omega_{m/2}^{m/2} = 1$, we can easily simplify the later terms as repetitions of the earlier terms.

The Fast Fourier Transform (FFT)

- Therefore we can write

$$\begin{aligned} P_A(\omega_m^k) &= P_{A^{[0]}}\left((\omega_m^k)^2\right) + \omega_m^k \cdot P_{A^{[1]}}\left((\omega_m^k)^2\right) \\ &= P_{A^{[0]}}\left(\omega_{m/2}^k\right) + \omega_m^k \cdot P_{A^{[1]}}\left(\omega_{m/2}^k\right). \end{aligned}$$

- Thus, we have reduced a problem of size m to two such problems of size $m/2$, plus a linear overhead to multiply the ω_m^k terms.

FFT algorithm

1: **function** $\mathrm{FFT}(A)$
2: $m \leftarrow \mathrm{length}(A)$ % assumed to be a power of 2
3: **if** $m = 1$ **then return** A
4: **else**
5: $A^{[0]} \leftarrow (A_0, A_2, \ldots A_{m-2})$;
6: $A^{[1]} \leftarrow (A_1, A_3, \ldots A_{m-1})$;
7: $y^{[0]} \leftarrow FFT(A^{[0]})$;
8: $y^{[1]} \leftarrow FFT(A^{[1]})$;
9: $\omega_m \leftarrow e^{i\frac{2\pi}{m}}$;

FFT algorithm

10: $\omega \leftarrow 1$; % a variable to hold powers of ω_m

11: **for** $k = 0$ to $k = m/2 - 1$ **do**:

$$\% \ \underbrace{P_A\left(\omega_m^k\right)}_{y_k} = \underbrace{P_{A^{[0]}}\left(\omega_{m/2}^k\right)}_{y_k^{[0]}} + \underbrace{\omega_m^k}_{\omega} \underbrace{P_{A^{[1]}}\left(\omega_{m/2}^k\right)}_{y_k^{[1]}}$$

12: $y_k \leftarrow y_k^{[0]} + \omega \cdot y_k^{[1]}$;

13: $\omega \leftarrow \omega \cdot \omega_m$;

14: **end for**

FFT algorithm

15: **for** $k = 0$ to $k = m/2 - 1$ **do**:

$$\% \ \underbrace{P_A\left(\omega_m^{m/2+k}\right)}_{y_{m/2+k}} = \underbrace{P_{A^{[0]}}\left(\omega_{m/2}^{k}\right)}_{y_k^{[0]}} + \underbrace{\omega_m^{m/2+k}}_{\omega} \underbrace{P_{A^{[1]}}\left(\omega_{m/2}^{k}\right)}_{y_k^{[1]}}$$

16: $y_{m/2+k} \leftarrow y_k^{[0]} + \omega \cdot y_k^{[1]}$;

17: $\omega \leftarrow \omega \cdot \omega_m$;

18: **end for**

19: **return** y

20: **end if**

21: **end function**

How fast is the Fast Fourier Transform?

- We have recursively reduced evaluation of a polynomial $P_A(x)$ with m coefficients at m roots of unity of order m to evaluations of two polynomials $P_{A^{[0]}}(y)$ and $P_{A^{[1]}}(y)$, each with $m/2$ coefficients, at $m/2$ many roots of unity of order $m/2$.

- Once we get these $m/2$ values of $P_{A^{[0]}}(y)$ and $P_{A^{[1]}}(y)$ we need m additional multiplications to obtain the values of

$$\underbrace{P_A\left(\omega_m^k\right)}_{y_k} = \underbrace{P_{A^{[0]}}\left(\omega_{m/2}^k\right)}_{y_k^{[0]}} + \omega_n^k \underbrace{P_{A^{[1]}}\left(\omega_{m/2}^k\right)}_{y_k^{[1]}}$$

for all $0 \leq k < m$.

How fast is the Fast Fourier Transform?

- Thus, we have reduced a problem of size m to two such problems of size $m/2$, plus a linear overhead.
- Consequently, our algorithm's run time satisfies the recurrence

$$T(m) = 2\,T\left(\frac{m}{2}\right) + c\,m.$$

- Case 2 of the Master Theorem gives $T(m) = \Theta(m \log m) = \Theta(n \log n)$.

Matrix representation of polynomial evaluation

- The evaluation of a polynomial $P_A(x) = A_0 + A_1 x + \ldots + A_{m-1} x^{m-1}$ at roots of unity ω_m^k of order m can be represented using the following matrix-vector product:

$$\underbrace{\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_m & \omega_m^2 & \cdots & \omega_m^{m-1} \\ 1 & \omega_m^2 & \omega_m^{2\cdot 2} & \cdots & \omega_m^{2\cdot(m-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_m^{m-1} & \omega_m^{2(m-1)} & \cdots & \omega_m^{(m-1)(m-1)} \end{pmatrix}}_{M} \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_{m-1} \end{pmatrix} = \begin{pmatrix} P_A(1) \\ P_A(\omega_m) \\ P_A(\omega_m^2) \\ \vdots \\ P_A(\omega_m^{m-1}) \end{pmatrix}$$

- The FFT is just a method of replacing this matrix-vector multiplication taking m^2 many multiplications with a $\Theta(m \log m)$ procedure.

Matrix representation of polynomial evaluation

- We also need a way to 'reconstruct' the coefficients from these values, i.e. perform the inverse DFT.

- Since M is a square Vandermonde matrix with distinct rows, it is invertible, so we can write:

$$\begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_{m-1} \end{pmatrix} = M^{-1} \begin{pmatrix} P_A(1) \\ P_A(\omega_m) \\ P_A(\omega_m^2) \\ \vdots \\ P_A(\omega_m^{m-1}) \end{pmatrix}.$$

- How do we find M^{-1}?

Another remarkable feature of the roots of unity

Claim

The inverse of matrix M is found by simply changing the signs of the exponents and dividing by m.

Let

$$Q = \frac{1}{m}\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_m^{-1} & \omega_m^{-2} & \dots & \omega_m^{-(m-1)} \\ 1 & \omega_m^{-2} & \omega_m^{-2\cdot 2} & \dots & \omega_m^{-2\cdot(m-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_m^{-(m-1)} & \omega_m^{-2(m-1)} & \dots & \omega_m^{-(m-1)(m-1)} \end{pmatrix}.$$

We need to prove that $MQ = I(= QM)$, where I is the identity matrix.

Another remarkable feature of the roots of unity

When we multiply MQ, the (i, j) entry in the product matrix is given by the product of the ith row of M and the jth column of Q:

$$(MQ)_{i,j} = \frac{1}{m} \begin{pmatrix} 1 & \omega_m^i & \omega_m^{2\cdot i} & \cdots & \omega_m^{i\cdot(m-1)} \end{pmatrix} \begin{pmatrix} 1 \\ \omega_m^{-j} \\ \omega_m^{-2j} \\ \vdots \\ \omega_m^{-(m-1)j} \end{pmatrix}$$

$$= \frac{1}{m} \sum_{k=0}^{m-1} \omega_m^{ik}\, \omega_m^{-jk} = \frac{1}{m} \sum_{k=0}^{m-1} \omega_m^{(i-j)k}$$

Another remarkable feature of the roots of unity

We have established that $(MQ)_{i,j} = \frac{1}{m}\sum_{k=0}^{m-1}\omega_m^{(i-j)k}$. We now have two possibilities:

1. if $i = j$, then

$$\begin{aligned}(MQ)_{i,j} &= \frac{1}{m}\sum_{k=0}^{m-1}\omega_m^0 \\ &= \frac{1}{m}\sum_{k=0}^{m-1}1 \\ &= \frac{1}{m}\cdot m = 1;\end{aligned}$$

Another remarkable feature of the roots of unity

2. if $i \neq j$, then $\sum_{k=0}^{m-1} \omega_m^{(i-j)k}$ represents the sum of a geometric progression with m terms and common ratio ω_m^{i-j}. Recalling that

$$1 + r + r^2 + \ldots + r^{m-1} = \frac{1 - r^m}{1 - r},$$

we have

$$\begin{aligned}
(MQ)_{i,j} &= \frac{1}{m} \cdot \frac{1 - \omega_m^{(i-j)m}}{1 - \omega_m^{i-j}} \\
&= \frac{1}{m} \cdot \frac{1 - (\omega_m^m)^{i-j}}{1 - \omega_m^{i-j}} \\
&= \frac{1}{m} \cdot \frac{1 - 1}{1 - \omega_m^{i-j}} = 0.
\end{aligned}$$

Another remarkable feature of the roots of unity

We have now proven that

$$(MQ)_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases},$$

that is, $MQ = I$. Therefore Q is the inverse of M, i.e.

$$M^{-1} = \frac{1}{m} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_m^{-1} & \omega_m^{-2} & \dots & \omega_m^{-(m-1)} \\ 1 & \omega_m^{-2} & \omega_m^{-2\cdot 2} & \dots & \omega_m^{-2\cdot(m-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_m^{-(m-1)} & \omega_m^{-2(m-1)} & \dots & \omega_m^{-(m-1)(m-1)} \end{pmatrix}.$$

Another remarkable feature of the roots of unity

So we get

$$\begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_{m-1} \end{pmatrix} = \frac{1}{m} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_m^{-1} & \omega_m^{-2} & \dots & \omega_m^{-(m-1)} \\ 1 & \omega_m^{-2} & \omega_m^{-2\cdot 2} & \dots & \omega_m^{-2\cdot(m-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_m^{-(m-1)} & \omega_m^{-2(m-1)} & \dots & \omega_m^{-(m-1)(m-1)} \end{pmatrix} \begin{pmatrix} P_A(1) \\ P_A(\omega_m) \\ P_A(\omega_m^2) \\ \vdots \\ P_A(\omega_m^{m-1}) \end{pmatrix}.$$

This is not so different to the original matrix-vector product!

The Inverse Fast Fourier Transform

The 'reconstruct' step requires us to convert from the sequence of values

$$\langle P_A(1), P_A(\omega_m), P_A(\omega_m^2), \ldots, P_A(\omega_m^{m-1})\rangle,$$

which we denoted by

$$\langle \widehat{A}_0, \widehat{A}_1, \widehat{A}_2, \ldots, \widehat{A}_{m-1}\rangle$$

back to the sequence of coefficients

$$\langle A_0, A_1, A_2, \ldots, A_{m-1}\rangle,$$

i.e. to compute the inverse DFT. We can use **the same** FFT algorithm with just two changes:

1. the root of unity ω_m is replaced by $\overline{\omega_m} = e^{-i\frac{2\pi}{m}}$,
2. the resulting output values are divided by m.

Inverse Fast Fourier Transform (IFFT)

1: **function** $\mathrm{IFFT}^*(\widehat{A})$
2: $m \leftarrow \mathrm{length}(\widehat{A})$
3: **if** $m = 1$ **then return** $\widehat{A}$
4: **else**
5: $\widehat{A}^{[0]} \leftarrow (\widehat{A}_0, \widehat{A}_2, \ldots \widehat{A}_{m-2})$;
6: $\widehat{A}^{[1]} \leftarrow (\widehat{A}_1, \widehat{A}_3, \ldots \widehat{A}_{m-1})$;
7: $y^{[0]} \leftarrow \mathit{IFFT}^*(\widehat{A}^{[0]})$;
8: $y^{[1]} \leftarrow \mathit{IFFT}^*(\widehat{A}^{[1]})$;
9: $\omega_m \leftarrow e^{-i\frac{2\pi}{m}}$; $\Leftarrow$ different from FFT

Inverse Fast Fourier Transform (IFFT)

10: $\omega \leftarrow 1$;
11: **for** $k = 0$ to $k = m/2 - 1$ **do**;
12: $y_k \leftarrow y_k^{[0]} + \omega \cdot y_k^{[1]}$;
13: $\omega \leftarrow \omega \cdot \omega_m$;
14: **end for**
15: **for** $k = 0$ to $k = m/2 - 1$ **do**;
16: $y_{m/2+k} \leftarrow y_k^{[0]} + \omega \cdot y_k^{[1]}$
17: $\omega \leftarrow \omega \cdot \omega_m$;
18: **end for**
19: **return** y;
20: **end if**
21: **end function**

Inverse Fast Fourier Transform (IFFT)

1: **function** $\mathrm{IFFT}(\widehat{A})$ $\Leftarrow$ different from FFT
2: **return** $IFFT^*(\widehat{A})/\mathrm{length}(\widehat{A})$
3: **end function**

Important note

Computer science books take the forward DFT operation to be the evaluation of the corresponding polynomial at all roots of unity $\omega_m^k = \exp(2\pi k\, i/m)$ and the inverse DFT to be the evaluation of the polynomial at the complex conjugates of the roots of unity, i.e. at $\omega_m^{-k} = \exp(-2\pi k\, i/m)$.

However, electrical engineering books do the opposite, with the direct DFT evaluating the polynomial at ω_m^{-k} and the inverse DFT at ω_m^k.

While for the purpose of multiplying polynomials both choices are equally good, the choice made by the electrical engineers is otherwise better. This will be explained in COMP4121 Advanced & Parallel Algorithms in the context of JPEG compression.

Important note

We did only the multiplication of polynomials, and did not apply it to multiplication of large integers. This is possible to do but one has to be careful because roots of unity are represented by floating point numbers. You have to show that if you do FFT with sufficient precision you can round off the results and obtain correct integer values, but all of this is tricky.

Earlier results along this line produced algorithms for multiplication of large (i.e. n-bit) integers which operate in time $\Theta(n \log n \log \log n)$ but very recently David Harvey of the School of Mathematics at UNSW came up with an algorithm to multiply large integers which runs in time $\Theta(n \log n)$.

Back to fast multiplication of polynomials

$$
\begin{array}{lcll}
P_A(x) & \xrightarrow[\Theta(n \log n)]{\text{DFT}} & \widehat{A} & \\
 & & \times & \\
P_B(x) & \xrightarrow[\Theta(n \log n)]{\text{DFT}} & \widehat{B} & \Theta(n) \\
 & & = & \\
P_C(x) & \xleftarrow[\Theta(n \log n)]{\text{IDFT}} & \widehat{C} &
\end{array}
$$

Back to fast multiplication of polynomials

- We can now compute the product $P_C(x) = P_A(x)\,P_B(x)$ of two polynomials $P_A(x)$ and $P_B(x)$ in time $\Theta(n \log n)$.

- Recall that the coefficients of $P_C(x)$ are given by

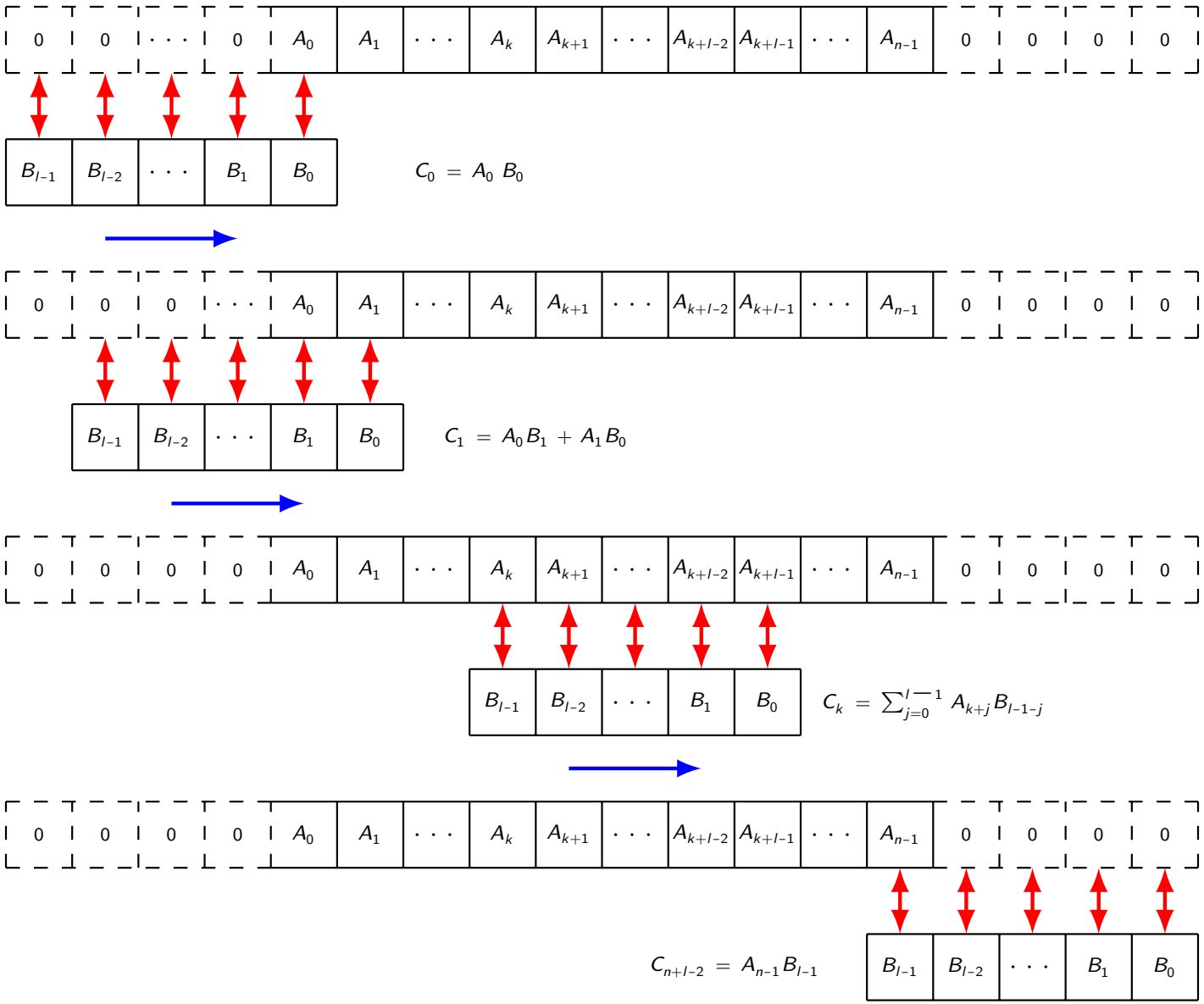
$$C_t = \sum_{i+j=t} A_i B_j.$$

  The coefficient sequence C is the convolution of the sequences A and B, denoted $A \star B$.

- So we can find the convolution of two n-term sequences in $\Theta(n \log n)$.

- Indeed, none of this requires that the two sequences be of the same length!

Visualizing Convolution $C = A \star B$



An Exercise

Problem

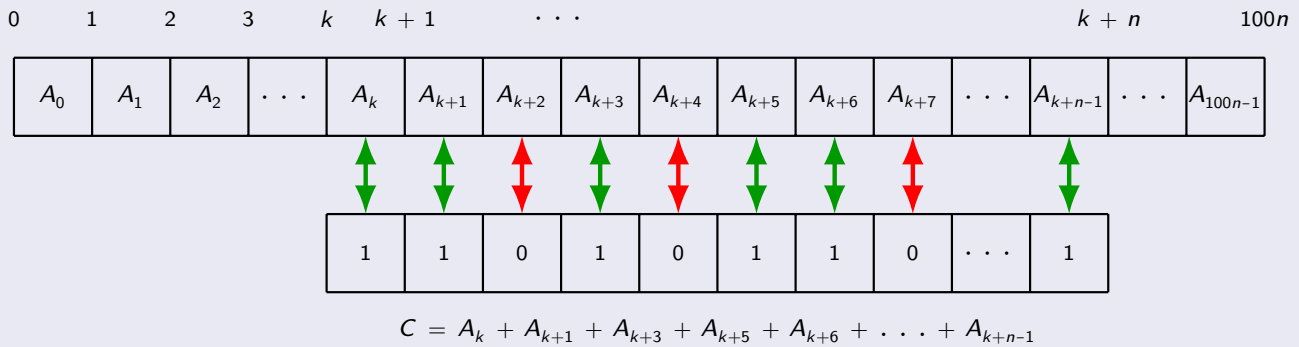
Suppose you are given a map of a straight sea shore of length $100n$ meters as a sequence on $100n$ numbers such that A_i is the number of fish between i^{th} meter of the shore and $(i+1)^{th}$ meter, $0 \leq i \leq 100n - 1$.

You also have a net of length n meters but unfortunately it has holes in it. The net is described as a sequence N of n ones and zeros, where 0's denote where the holes are.

If you throw the net starting at meter k and ending at meter $k + n$, then you will catch only the fish in one meter stretches of the shore where the corresponding bit of the net is 1.

An Exercise

Problem (continued)



Find the spot where you should place the left end of your net in order to catch the largest possible number of fish using an algorithm which runs in time $O(n \log n)$.

Hint

Let N' be the net sequence N in the reverse order. Compute $A \star N'$ and look for the peak of that sequence.

Table of Contents

PUZZLE!!

On a circular highway there are n petrol stations, unevenly spaced, each containing a different quantity of petrol. It is known that the total quantity of petrol on all stations is enough to go around the highway once, and that the tank of your car can hold enough fuel to make a trip around the highway.

Prove that there always exists a station among all of the stations on the highway, such that if you take it as a starting point and take the fuel from that station, you can continue to make a complete round trip around the highway, never emptying your tank before reaching the next station to refuel.

That's All, Folks!!