Note: Your TA may not get to all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. The discussion worksheet is also a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Squaring vs multiplying: matrices

The square of a matrix A is its product with itself, AA.

(a) Show that five multiplications are sufficient to compute the square of a 2×2 matrix.

(b) What is wrong with the following algorithm for computing the square of an $n \times n$ matrix?

"Use a divide-and-conquer approach as in Strassen's algorithm, except that instead of getting 7 subproblems of size $n/2$, we now get 5 subproblems of size $n/2$ thanks to part (a). Using the same analysis as in Strassen's algorithm, we can conclude that the algorithm runs in $\mathcal{O}(n^{\log_2 5})$ time."

(c) In fact, squaring matrices is no easier than multiplying them. Show that if $n \times n$ matrices can be squared in $\Theta(n^c)$ time, then any $n \times n$ matrices can be multiplied in $\Theta(n^c)$ time.

(Hint: Given matrices X, Y, is there some matrix A such that we can easily compute XY given A^2?)

Solution:

a)

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^2 = \begin{bmatrix} a^2 + bc & b(a+d) \\ c(a+d) & bc + d^2 \end{bmatrix}$$

Hence the 5 multiplications $a^2, d^2, bc, b(a+d)$ and $c(a+d)$ suffice to compute the square.

b) We have:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^2 = \begin{bmatrix} A^2 + BC & AB + BD \\ CA + DC & CB + D^2 \end{bmatrix} \neq \begin{bmatrix} A^2 + BC & B(A+D) \\ C(A+D) & BC + D^2 \end{bmatrix}.$$

We end up getting 5 subproblems that are *not of the same type as the original problem*: We started with a squaring problem for a matrix of size $n \times n$ and three of the 5 subproblems now involve *multiplying* $n/2 \times n/2$ matrices. Hence the recurrence $T(n) = 5T(n/2) + O(n^2)$ does not make sense.

(Also, note that matrices don't commute! That is, in general $BC \neq CB$, so we cannot reuse that computation)

c) Given two $n \times n$ matrices X and Y, create the $2n \times 2n$ matrix A:

$$A = \begin{bmatrix} 0 & X \\ Y & 0 \end{bmatrix}$$

It now suffices to compute A^2, as its upper left block will contain XY:

$$A^2 = \begin{bmatrix} XY & 0 \\ 0 & YX \end{bmatrix}$$

So if we can compute A^2 in time $\Theta(n^c)$, then we can also compute XY in time $\Theta(n^c)$ - the asymptotic runtimes are the same because A's dimensions are only a constant larger than X and Y's dimensions.

Note: This is an example of a reduction, and is an important concept that we will see over and over again in this course. We are saying that matrix squaring is no easier than matrix multiplication — because we can trick any program for matrix squaring to actually solve the more general problem of matrix multiplication.

2 Complex numbers review

A *complex number* is a number that can be written in the rectangular form $a + bi$ (i is the imaginary unit, with $i^2 = -1$). The following famous equation (*Euler's formula*) relates the polar form of complex numbers to the rectangular form:

$$re^{i\theta} = r(\cos\theta + i\sin\theta)$$

In polar form, $r \geq 0$ represents the distance of the complex number from 0, and θ represents its angle. Note that since $\sin(\theta) = \sin(\theta + 2\pi), \cos(\theta) = \cos(\theta + 2\pi)$, we have $re^{i\theta} = re^{i(\theta+2\pi)}$ for any r, θ.

The n-th *roots of unity* are the n complex numbers satisfying $\omega^n = 1$. They are given by

$$\omega_k = e^{2\pi ik/n}, \qquad k = 0, 1, 2, \dots, n-1$$

(a) Let $x = e^{2\pi i3/10}, y = e^{2\pi i5/10}$ which are two 10-th roots of unity. Compute the product $x \cdot y$. Is this an n-th root of unity for some n? Is it a 10-th root of unity?

What happens if $x = e^{2\pi i6/10}, y = e^{2\pi i7/10}$?

Solution: $x \cdot y = e^{2\pi i8/10}$. This is always an 10-th root of unity (it is in general). But because $8/10 = 4/5$, this is also a 5th root of unity.

If $x = e^{2\pi i6/10}, y = e^{2\pi i7/10}$, then we ‘wind around‘ and the product becomes $e^{2\pi i13/10} = e^{2\pi i3/10}$.

(b) Show that for any n-th root of unity $\omega \neq 1$, $\sum_{k=0}^{n-1} \omega^k = 0$, when $n > 1$.

Hint: Use the formula for the sum of a geometric series $\sum_{k=0}^{n} \alpha^k = \frac{\alpha^{n+1}-1}{\alpha-1}$. It works for complex numbers too!

Solution: Remember that $\omega^n = 1$. So
$\sum_{k=0}^{n-1} \omega^k = \frac{\omega^n-1}{\omega-1} = \frac{1-1}{\omega-1} = 0$

(c) (i) Find all ω such that $\omega^2 = -1$.

Solution: $\omega = i, -i$

There are many ways to arrive at the solution, here's one: Squaring both sides, we get $\omega^4 = 1$. So we only need to consider the 4th roots of unity, $e^{2\pi i\cdot 0/4}, e^{2\pi i\cdot 1/4}, e^{2\pi i\cdot 2/4}, e^{2\pi i\cdot 3/4}$, or equivalently $1, i, -1, -i$. Geometrically, we get these by going $0, 1, 2, 3$ quarters of the way around the complex unit circle. Of these four values, the ones that square to -1 are $i, -i$.

(ii) Find all ω such that $\omega^4 = -1$.

Solution: $\omega = e^{2\pi i\cdot 1/8}, e^{2\pi i\cdot 3/8}, e^{2\pi i\cdot 5/8}, e^{2\pi i\cdot 7/8}$

Similarly to the previous part, squaring both sides we get $\omega^8 = 1$, so we only need to consider the 8th roots of unity. However, $\omega^4 \neq 1$, so ω is not a 4th root of unity. The 8th roots of unity that are not 4th roots of unity are $e^{2\pi i\cdot 1/8}, e^{2\pi i\cdot 3/8}, e^{2\pi i\cdot 5/8}, e^{2\pi i\cdot 7/8}$, and we can check that these all are solutions to $\omega^4 = -1$.

3 FFT Intro

We will use ω_n to denote the first n-th root of unity $\omega_n = e^{2\pi i/n}$. The most important fact about roots of unity for our purposes is that the squares of the $2n$-th roots of unity *are* the n-th roots of unity.

Fast Fourier Transform! The *Fast Fourier Transform* $\text{FFT}(p, n)$ takes arguments n, some power of 2, and p is some vector $[p_0, p_1, \ldots, p_{n-1}]$.

Treating p as a polynomial $P(x) = p_0 + p_1 x + \ldots + p_{n-1}x^{n-1}$, the FFT computes the value of $P(x)$ for all x that are n-th roots of unity by doing the following matrix multiplication in $\mathcal{O}(n \log n)$ time:

$$\begin{bmatrix} P(1) \\ P(\omega_n) \\ P(\omega_n^2) \\ \vdots \\ P(\omega_n^{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{(n-1)} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{(n-1)} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{bmatrix} \cdot \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix}$$

If we let $E(x) = p_0 + p_2 x + \ldots p_{n-2}x^{n/2-1}$ and $O(x) = p_1 + p_3 x + \ldots p_{n-1}x^{n/2-1}$, then $P(x) = E(x^2) + xO(x^2)$, and then $FFT(p, n)$ can be expressed as a divide-and-conquer algorithm:

1. Compute $E' = \text{FFT}(E, n/2)$ and $O' = \text{FFT}(O, n/2)$.
2. For $i = 0 \ldots n-1$, assign $P(\omega_n^i) \leftarrow E((\omega_n^i)^2) + \omega_n^i O((\omega_n^i)^2)$

Also observe that:

$$\frac{1}{n}\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \cdots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-4} & \cdots & \omega_n^{-2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \cdots & \omega_n^{-(n-1)(n-1)} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{(n-1)} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{(n-1)} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{bmatrix}^{-1}$$

(You should verify this on your own!) And so given the values $P(1), P(\omega_n), P(\omega_n^2)\ldots$, we can compute P by doing the following matrix multiplication:

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix} = \frac{1}{n}\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \cdots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-4} & \cdots & \omega_n^{-2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \cdots & \omega_n^{-(n-1)(n-1)} \end{bmatrix} \cdot \begin{bmatrix} P(1) \\ P(\omega_n) \\ P(\omega_n^2) \\ \vdots \\ P(\omega_n^{n-1}) \end{bmatrix}$$

This can be done in $O(n \log n)$ time using a similar divide and conquer algorithm.

(a) Let $p = [p_0]$. What is $\text{FFT}(p, 1)$?

Solution: Notice the FFT matrix is just $[1]$, so $\text{FFT}(p, 1) = [p_0]$.

(b) Use the FFT algorithm to compute $\text{FFT}([1, 4], 2)$ and $\text{FFT}([3, 2], 2)$.

Solution: $\text{FFT}([1, 4], 2) = [5, -3]$ and $\text{FFT}([3, 2], 2) = [5, 1]$.

We show how to compute $\text{FFT}([1, 4], 2)$, and $\text{FFT}([3, 2], 2)$ is similar.

First we compute $\text{FFT}([1], 1) = [1] = E'$ and $\text{FFT}([4], 1) = [4] = O'$ by part (a). Notice that $E' = [E(1)] = 1$ and $O' = [O(1)] = [4]$, so when we need to use these values later they have already been computed in E' and O'.

Let P be our result. We wish to compute $P(\omega_2^0) = P(1)$ and $P(\omega_2^1) = P(-1)$.

$$P(1) = E(1) + 1 \cdot O(1) = 1 + 4 = 5$$
$$P(-1) = E(1) + -1 \cdot O(1) = 1 - 4 = -3$$

So our answer is $[5, -3]$.

(c) Use your answers to the previous parts to compute $\text{FFT}([1, 3, 4, 2], 4)$.

Solution: $\omega_4 = i$. The following table is good to keep handy:

$\boldsymbol{\omega_4^i}$	1	i	-1	$-i$
$\boldsymbol{(\omega_4^i)^2}$	1	-1	1	-1

Let $E' = \text{FFT}([1, 4], 2) = [5, -3]$ and $O' = \text{FFT}([3, 2], 2) = [5, 1]$. Notice that $E' = [E(1), E(-1)] = [5, -3] =$ and $O' = [O(1), O(-1)] = [5, 1]$, so when we need to use these values later they have already been computed in the divide step. Let R be our result, we wish to compute $R(1), R(i), R(-1), R(-i)$.

$$R(1) = E(1) + 1 \cdot O(1) = 5 + 5 = 10$$
$$R(i) = E(-1) + i \cdot O(-1) = -3 + i$$
$$R(-1) = E(1) - 1 \cdot O(1) = 5 - 5 = 0$$
$$R(-i) = E(-1) - i \cdot O(-1) = -3 - i$$

So our answer $[10, -3 + i, 0, -3 - i]$.

(d) Describe how to multiply two polynomials $p(x), q(x)$ in coefficient form of degree at most d.

Solution: The idea is to take the FFT of both p and q, multiply the evaluations, and then take the inverse FFT. Note that $p \cdot q$ has degree at most $2d$, which means we need to pick n as the smallest power of 2 greater than $2d$, call this 2^k. We can zero-pad both polynomials so they have degree $2^k - 1$.

Then $M = \text{FFT}(p, 2^k) \cdot \text{FFT}(q, 2^k)$ (with multiplication elementwise) computes $pq(\omega_{2^k}^i)$ for all $i = 0, \ldots, 2^k - 1$.

We take the inverse FFT of M to get back to $p \cdot q$ in coefficient form.

4 Practice with FFT

What is the FFT of $(1, 0, 0, 0)$? What is the appropriate value of ω in this case? And of which sequence is $(1, 0, 0, 0)$ the FFT?

Solution: Using $\omega = i$, we have

$$FFT(1, 0, 0, 0) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

If $\text{FFT}(x) = (1, 0, 0, 0)$, then

$$x = \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix}.$$