Lecture 2

Claim: There is a unique polynomial of degree $n-1$ that takes the following point-value pairs:
$$(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1}),$$
where the x_i's are distinct.

$$\underbrace{\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ & & \vdots & & \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix}} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

call this matrix V: this is a Vandermonde matrix.

So $\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = V^{-1} \cdot \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix}$. We need to show that when the x_i's are distinct, the matrix V is invertible.

Subclaim. $\det(V) = \prod_{0 \le i < j \le n-1} (x_j - x_i)$.

Proof. We would like to come up with an upper triangular matrix $\mathcal{U}$ such that $M\mathcal{U} = \mathcal{L}$ where $\mathcal{L}$ is the following lower triangular matrix.

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ * & d_1 & 0 \cdots & 0 \\ * & * & d_2 \cdots & 0 \\ * & * & \cdots & d_{n-1} \end{bmatrix}$$

where
$$d_1 = (x_1 - x_0)$$
$$d_2 = (x_2 - x_1)(x_2 - x_0)$$
$$d_3 = (x_3 - x_2)(x_3 - x_1)(x_3 - x_0)$$
$$\vdots$$
$$d_{n-1} = \prod_{i=0}^{n-2} (x_{n-1} - x_i)$$

The $*$'s are values we do not care about.

Since determinant is a multiplicative function, we have $\det(V) \cdot \det(\mathcal{U}) = \det(\mathcal{L})$

- The matrix $\mathcal{U}$ will have 1's on its diagonal. Recall that this is an upper triangular matrix. So $\det(\mathcal{U}) = 1$.

Hence $\det(V) = \det(\mathcal{L}) = \prod_{0 \le i < j \le n-1} (x_j - x_i)$.

- What is the matrix $\mathcal{U}$? Any guesses?

Let $\mathcal{U}$ be the matrix
$$\begin{bmatrix} 1 & * & * & \\ 0 & 1 & * & \\ \vdots & 0 & 1 & \cdots \\ 0 & \vdots & \vdots & \\ & 0 & 0 & \end{bmatrix}$$

the constant polynomial 1

these are the coefficients of the polynomial $x - x_0$: 1 is the coeff. of x and $*$ is $-x_0$

these are the coeff. of the poly. $(x-x_0)(x-x_1)$ where 1 is the coeff of x^2 the $*$ above 1 is $-(x_0+x_1)$: this is the coeff. of x, and the top $*$ is the const. term $x_0 x_1$

Check that pre-multiplying $\mathcal{U}$ with the row vector $[1 \;\; x_i \;\; x_i^2 \;\; \ldots \;\; x_i^{n-1}]$ produces the vector $[1 \;\; (x_i - x_0) \;\; (x_i-x_0)\cdot(x_i-x_1) \;\; \ldots \;\; \underbrace{0 \;\ldots\; 0}_{n-1-i \text{ 0's here}}]$

Thus $V\mathcal{U} = \mathcal{L}$. □

Recall that we want to design a fast algorithm to multiply 2 degree $(n-1)$ polynomials $A(x)$ and $B(x)$ — these are given in coefficient form.

We considered the following approach:

Step 1: Convert $A(x)$ and $B(x)$ to point-value representation.

Step 2: Obtain their product $P(x)$ in point-value representation.

Step 3: Convert $P(x)$ back to coefficient form.

Let us focus on Step 1. Here we want to evaluate $A(x)$ at $2n-1$ points and also evaluate $B(x)$ at the same $2n-1$ points.

Evaluating $A(x)$ at some point $x = x_0$ takes $\Theta(n)$ time by Horner's rule.

- However we do not wish to spend $\Theta(n^2)$ time to evaluate $A(x)$ at $2n-1$ points.

Say we want to evaluate $A(x)$ at 2 points α and β, i.e., we want to compute $A(\alpha)$ and $A(\beta)$.

What α and β should we choose so that computing $A(\alpha)$ and $A(\beta)$ are not independent tasks and we can share work?

Take $\beta = -\alpha$. Let $\alpha = 1$ and $\beta = -1$.

$$A(x) = A_0(x^2) + x \cdot A_1(x^2)$$

where
$$A_0(x^2) = a_0 + a_2 x^2 + a_4 x^4 + \ldots$$
$$A_1(x^2) = a_1 + a_3 x^2 + a_5 x^4 + \ldots$$

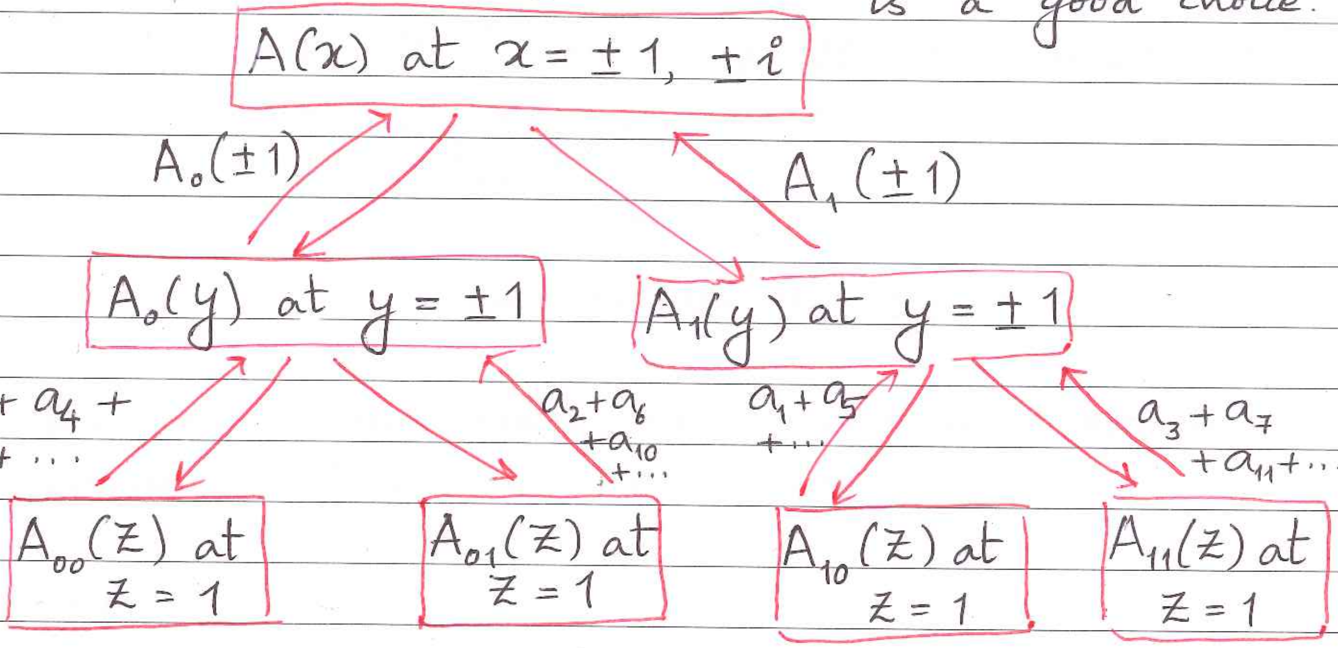
$$A(1) = A_0(1) + A_1(1)$$
$$A(-1) = A_0(1) - A_1(1)$$

Observation 1. Evaluating a polynomial at ± 1 is a good choice.

Suppose we want to evaluate $A(x)$ at 4 points
- what 4 points should we choose so that evaluating $A(x)$ at these 4 points allows us to share work?

Using Observation 1, evaluating $A_0(\pm 1)$, $A_1(\pm 1)$ is a good choice.
- this means evaluate $A(x)$ at $x = \pm 1, \pm i$.

Observation 2. Evaluating a polynomial at $\pm 1, \pm i$ is a good choice.



Suppose we want to evaluate $A(x)$ at 8 points:
- take $x = e^{\frac{2\pi i}{8} \cdot k}$ for $k = 0, \dots, 7$

$$= \cos \frac{k\pi}{4} + i \sin \frac{k\pi}{4} \text{ for } k = 0, \dots, 7$$

Going back to our starting problem:

- we want to evaluate $A(x) = a_0 + \dots + a_{n-1} x^{n-1}$ at $2n-1$ different points

First, let $N \geq 2n-1$ be the smallest power of 2 that is $\geq 2n-1$.

Second, add "0" coefficients to $A(x)$ to make it of the form $\sum_{j=0}^{N-1} a_j x^j$

- so A has N terms
- we want to evaluate $A(x)$ at N distinct points $\alpha_0, \alpha_1, \dots, \alpha_{N-1}$
- take $\alpha_0, \alpha_1, \dots, \alpha_{N-1}$ as the N roots of 1, i.e., the zeros of the polynomial $x^N - 1$
  * these are the complex numbers $\underbrace{e^{\frac{2\pi i}{N}\cdot k} \quad \text{for } k = 0, 1, \dots, N-1}_{\text{these are denoted by } 1, \omega, \omega^2, \dots, \omega^{N-1}}$

Let us now write down our recursive algorithm to evaluate $A(x)$ at $x = 1, \omega, \omega^2, \dots, \omega^{N-1}$

Eval(A, N)

1. if $N = 1$ then return a_0
2. else
   - 2.1 write $A(x) = A_0(x^2) + x \cdot A_1(x^2)$
   - 2.2 call Eval(A_0, $N/2$)
     let $\langle u_0, u_1, \dots, u_{N/2-1} \rangle$ be the ordered tuple returned.
   - 2.3 call Eval(A_1, $N/2$)
     let $\langle v_0, v_1, \dots, v_{N/2-1} \rangle$ be the ordered tuple returned.

[note that
$u_0 = A_0(1)$, $u_1 = A_0(\omega^2)$, …; similarly $v_0 = A_1(1)$, $v_1 = A_1(\omega^2)$, …

2.4 $A(1) = A_0(1) + A_1(1)$

$= u_0 + v_0$

Date

$$A(\omega) = A_0(\omega^2) + \omega \cdot A_1(\omega^2)$$
$$= u_1 + \omega \cdot v_1$$
$$\vdots$$
$$A(\omega^{N/2}) = A_0(1) + \omega^{N/2} \cdot A_1(1)$$
$$= u_0 - v_0$$
$$\vdots$$
$$A(\omega^{N-1}) = A_0(\omega^{N-2}) + \omega^{N-1} \cdot A_1(\omega^{N-2})$$
$$= u_{N/2-1} + \omega^{N-1} \cdot v_{N/2-1}$$

2.5 Return $\langle A(1), A(\omega), \ldots, A(\omega^{N-1}) \rangle$.

What is the running time of Eval(A, N)?

Observe that
$$T(N) = 2T(N/2) + cN \quad \text{for } N \geq 2$$
$$= O(1) \quad \text{for } N = 1$$

where $T(N)$ is the running time of Eval(A, N); c is a constant here.

This solves to $T(N) = O(N \log N)$.

What we have accomplished is the following multiplication:

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega & \cdots & \omega^{N-1} \\ & & \vdots & \\ 1 & \omega^{N-1} & \cdots & \omega^{(N-1)^2} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{bmatrix} = \begin{bmatrix} A(1) \\ A(\omega) \\ \vdots \\ A(\omega^{N-1}) \end{bmatrix}$$

This is called the Discrete Fourier Transform (DFT)

Our algorithm is called Fast Fourier Transform or FFT.