https://www.youtube.com/watch?v=h7apO7q16V0

Algorithm Design & Analysis (CSE222)

Lecture-6

Recap

- Select k Smallest Elements
    - Deterministic Algorithm: Partition()

$\textsc{DetQuickSelect}(A, k)$

if $(|A| == 1)$
    Return A
$p = \textsc{ChoosePivot}(A)$
$Lesser = \{A[i] : A[i] < p\}$; $Greater = \{A[i] : A[i] > p\}$
if $(|Lesser| == k - 1)$
    Return p
if $(|Lesser| > k - 1)$
    Return $\textsc{DetQuickSelect}(Lesser, k)$
else
    Return $\textsc{DetQuickSelect}(Greater, k - |Lesser| - 1)$

$\textsc{ChoosePivot}(A)$

Initialize an empty array C of size $|A|/5$; $i = 1$
Split A into $|A|/5$ groups as $B_1, \ldots B_g$ where $g = |A|/5$.
for $j = 1 \cdots g$
    $q = Median(B_j)$
    $C[i] = q$; $i = i + 1$
$p = \textsc{DetQuickSelect}(C, |C|/2)$
Return p

Recap

- Select k Smallest Elements
  - Deterministic Algorithm: Partition()
- Fast Fourier Transform
  - Convolution

Outline

- Fast Fourier Transform (Contd.)

- Dynamic Programming

Polynomial Multiplication (Convolution)

Given, two polynomials $P(x)$ and $Q(x)$ compute $R(x) = P(x) \cdot Q(x)$.

Let, $P(x) = 1 + 2x^2$ and $Q(x) = 2x + x^2$ then

$R(x) = 2x + x^2 + 4x^3 + 2x^4$.

PolyMult(P, Q):
Compute $R(x) = P(x) \cdot Q(x)$.

R = [0, 2, 1, 4, 2]

Coefficient Value Representation.

Polynomial Multiplication (Convolution)

PolyMult(P, Q):
Compute $R(x) = P(x) \cdot Q(x)$.

If $P(x)$ and $Q(x)$ are degree d polynomials then PolyMult(P, Q) will take $O(d^2)$ time.

for $i = 1 \dots d$
  for $j = 1 \dots d$
    $R[i+j] = R[i+j] + P[i] \cdot Q[j]$

Polynomial Multiplication (Convolution)

How many points do we need to represent a polynomial of degree 1?

Degree 1 polynomial is a line, so two points enough.

$P(x) = p_0 + p_1 \cdot x$.

If (3, 0) & (0, 3) are two points then $P(x) = 3 - x$. (**Point Value Representation**)

Claim: Any d degree polynomial can be uniquely represented by d + 1 points.

Proof

Claim: Any d degree polynomial can be uniquely represented by d + 1 points.

$$\{(x_0, P(x_0)), (x_1, P(x_1)), \ldots, (x_d, P(x_d))\}$$

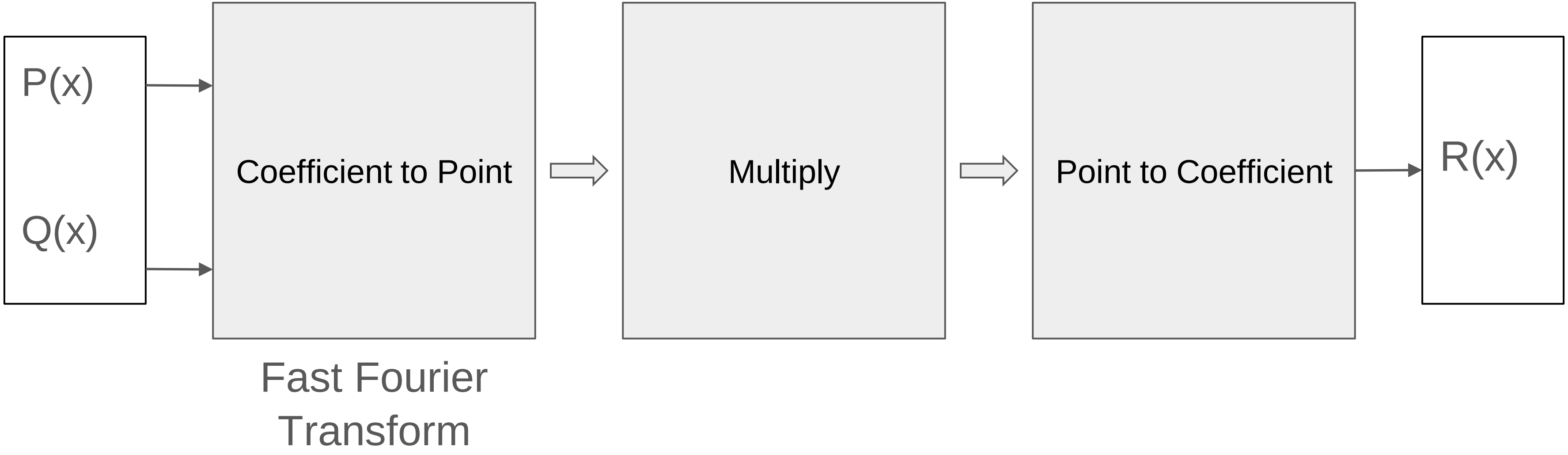
$$P(x) = p_0 + p_1 x + p_2 x^2 + \cdots + p_d x^d$$

$$\begin{bmatrix} P(x_0) \\ P(x_1) \\ \vdots \\ P(x_d) \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^d \\ 1 & x_1 & x_1^2 & \cdots & x_1^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_d & x_d^2 & \cdots & x_d^d \end{bmatrix}}_{M} \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_d \end{bmatrix}$$

- M is invertible for unique $x_0, \ldots, x_d$
- So, unique $p_0, \ldots, p_d$, i.e., $p = M^{-1} \cdot y$
- So, unique polynomial.

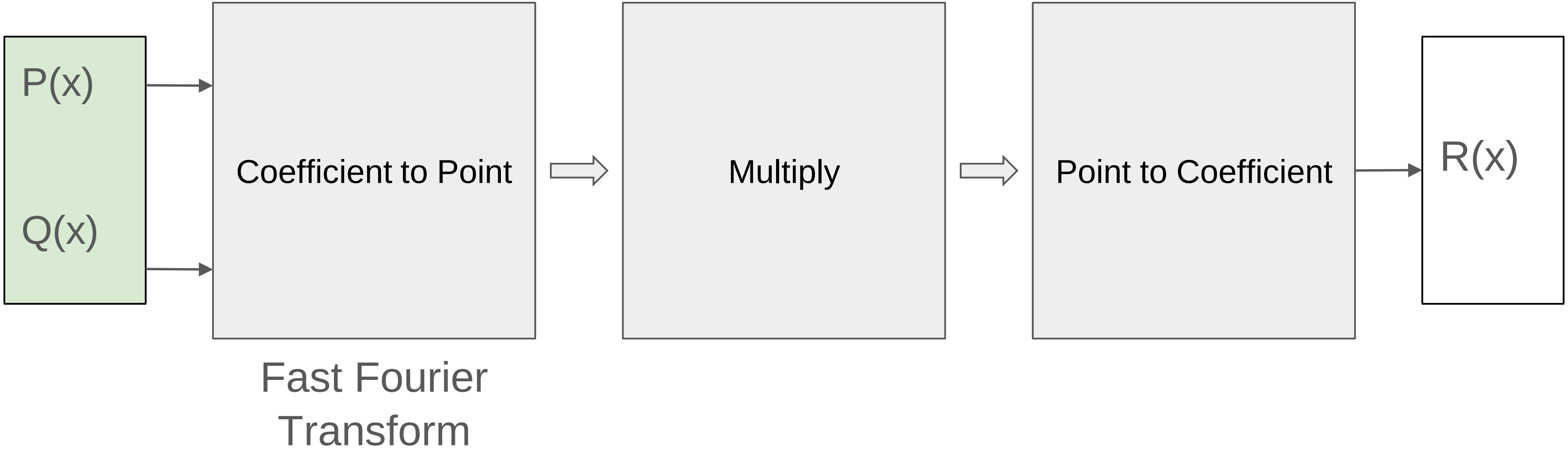
Polynomial Multiplication (Convolution)

Given point value representation of P(x) and Q(x) of degree d. Propose an efficient algorithm to multiply P(x) and Q(x).



Polynomial Multiplication (Convolution)

Given point value representation of P(x) and Q(x) of degree d. Propose an efficient algorithm to multiply P(x) and Q(x).



Fast Fourier Transform

Consider the polynomial $P(x) = p_0 + p_1 x + p_2 x^2 + \cdots + p_d x^d$

Coefficient to Point

Compute n ≥ d+1 pairs of (point, value) for P(x).

$$\begin{array}{c} (1, P(1)) \\ (2, P(2)) \\ \vdots \\ (n, P(n)) \end{array} \longrightarrow \begin{array}{c} (1, p_0 + p_1 \cdot 1 + p_2 \cdot 1^2 + \cdots + p_d \cdot 1^d) \\ (2, p_0 + p_1 \cdot 2 + p_2 \cdot 2^2 + \cdots + p_d \cdot 2^d)) \\ \vdots \\ (n, p_0 + p_1 \cdot n + p_2 \cdot n^2 + \cdots + p_d \cdot n^d)) \end{array}$$

P(x) at unique n points.

Running time: O(nd) ~ O(d^2) !!!

What if, we only need to compute n/2 (point, value) pairs and use it to get n pairs.

Coefficient to Point

Let $P(x) = x^2$ and $n = 4$.

(1,1)	&	(-1,1)	$P(-x) = P(x)$
(2,4)	&	(-2,4)	

Let $P(x) = x^3$ and $n = 4$.

(1,1)	&	(-1,-1)	$P(-x) = -P(x)$
(2,8)	&	(-2,-8)	

Coefficient to Point

General function $P(x) = 3x^5 + 2x^4 + x^3 + 7x^2 + 5x + 1$

Calculate P(x) at $\pm x_1, \pm x_2, \ldots, \pm x_{n/2}$

$$P(x) = (2x^4 + 7x^2 + 1) + (3x^5 + x^3 + 5x)$$

$$P(x) = \underbrace{(2x^4 + 7x^2 + 1)}_{P_e(x^2)} + x\underbrace{(3x^4 + x^2 + 5)}_{P_o(x^2)}$$

$$P(x) = P_e(x^2) + xP_o(x^2)$$

$$P(x_i) = P_e(x_i^2) + x_iP_o(x_i^2)$$

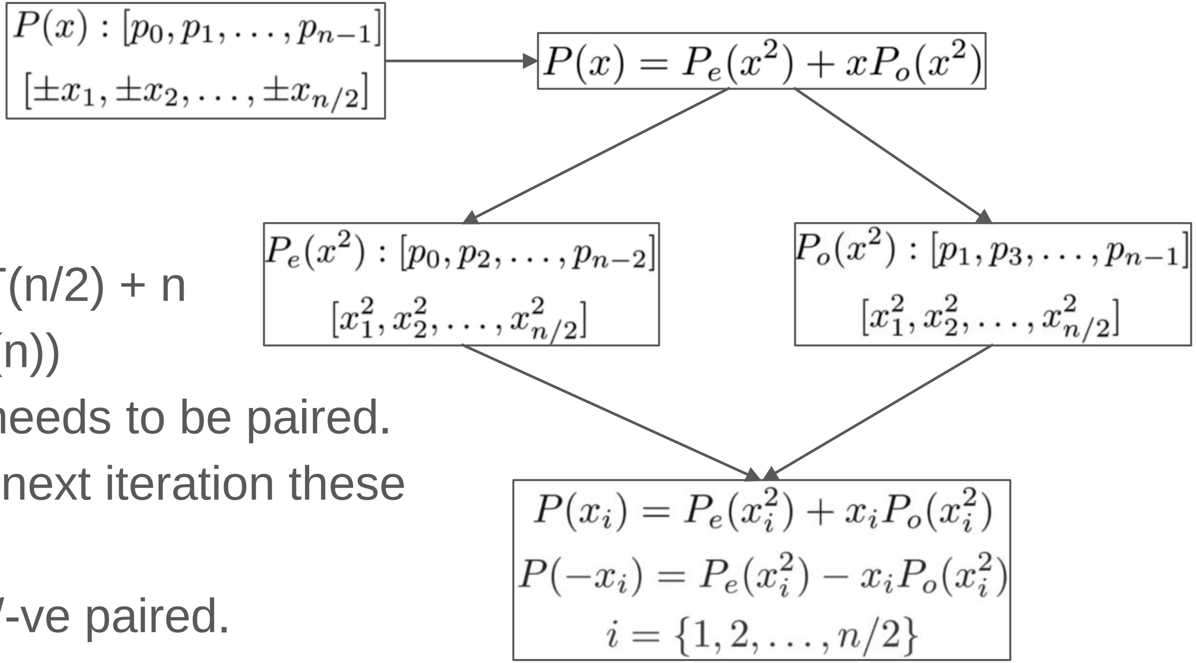
$$P(-x_i) = P_e(x_i^2) - x_iP_o(x_i^2)$$

Observations

- $P_e(x^2)$ and $P_o(x^2)$ are of degree 2, even though P(x) was a 5 degree polynomial.
- Calculate $P_e(x^2)$ and $P_o(x^2)$ at $x_1^2, x_2^2, \ldots, x_{n/2}^2$ points.

Coefficient to Point

I/P: P = $[p_0, p_1, \ldots, p_{n-1}]$
O/P: $[P(x_0), \ldots, P(X_{n-1})]$

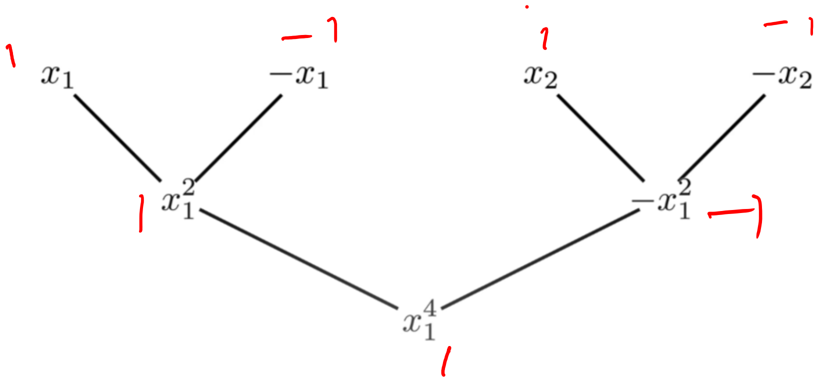


<u>Observations</u>

- Recurrence: T(n) = 2T(n/2) + n
- Running time O(n·log(n))
- $[\pm x_1, \pm x_2, \ldots, \pm x_{n/2}]$ needs to be paired.
- So, $[x_1^2, x_2^2, \ldots, x_{n/2}^2]$. For next iteration these needs to paired.
- But these are not +ve/-ve paired.
- Expand the domain of initial points, i.e., $[\pm x_1, \pm x_2, \ldots, \pm x_{n/2}]$ complex numbers.

Coefficient to Point

Let P(x) = $x^3 - x^2 + x - 1$ & n = 4.

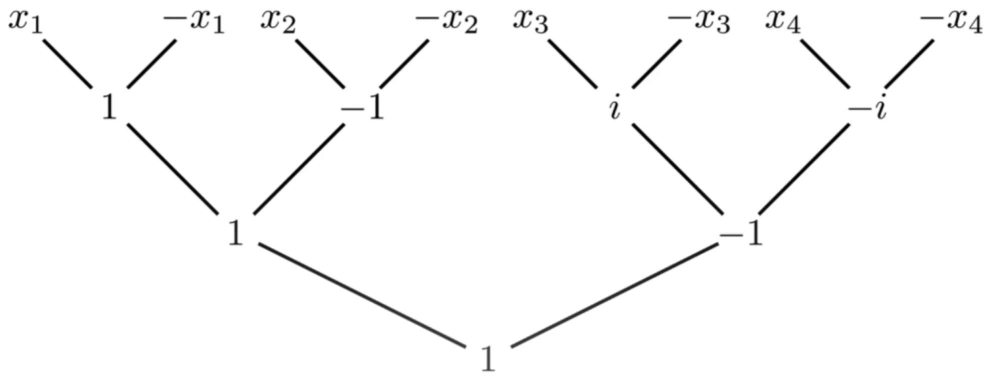


Let $x_1 = 1$.

So, our initial points are solution to $x^4 = 1$.

Coefficient to Point

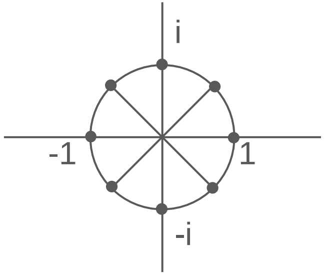
Let $P(x) = x^6 - x^5 + x^4 - x^3 + x^2 - x + 1$ & $n = 8$ (>7).



So, our initial points are solution to $x^8 = 1$.

Coefficient to Point

Solving $x^n = 1$ gives,



$e^{i\theta} = \cos(\theta) + i \cdot \sin(\theta)$

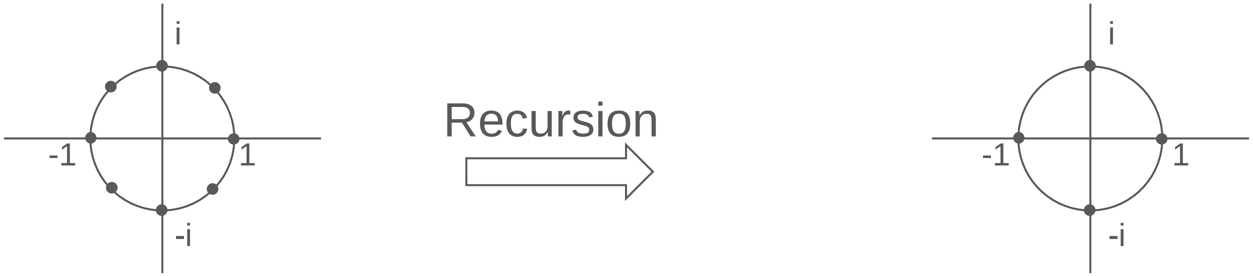
$$\omega = e^{\frac{2\pi i}{n}}$$

Roots are $[\omega^0, \omega^1, \ldots, \omega^{n-1}]$.

Notice, $\omega^{j + n/2} = -\omega^j$ in pairs!

Coefficient to Point

Squaring n roots of unity results.



Evaluate p(x) at [$1, \omega^1, \ldots, \omega^{n-1}$]

Evaluate $P_e(x^2)$ and $P_e(x^2)$ at [$1, \omega^2, \ldots, \omega^{2(n/2-1)}$]

These are also paired.

Coefficient to Point

Input: P = [p_0, p_1, …, p_{n-1}]
Output y = [P(ω^0), P(ω^1), …, P(ω^{n-1})]

$$P(x) : [p_0, p_1, \ldots, p_{n-1}]$$

$$\omega = e^{\frac{2\pi i}{n}} : [\omega^0, \omega^1, \ldots, \omega^{n-1}]$$

Base: n = 1, P(1).

$$P_e(x^2) : [p_0, p_2, \ldots, p_{n-2}] \qquad P_o(x^2) : [p_1, p_3, \ldots, p_{n-1}]$$

$$[\omega^0, \omega^2, \ldots, \omega^{n-2}] \qquad [\omega^0, \omega^2, \ldots, \omega^{n-2}]$$

$$P(\omega^j) = P_e(\omega^{2j}) + \omega^j P_o(\omega^{2j})$$

$$P(-\omega^j) = P_e(\omega^{2j}) - \omega^j P_o(\omega^{2j})$$

$$j \in \{0, 1, \ldots (n/2 - 1)\}$$

Coefficient to Point

Input: P = $[p_0, p_1, \ldots, p_{n-1}]$
Output y = $[P(\omega^0), P(\omega^1), \ldots, P(\omega^{n-1})]$

$$P(x) : [p_0, p_1, \ldots, p_{n-1}]$$

$$\omega = e^{\frac{2\pi i}{n}} : [\omega^0, \omega^1, \ldots, \omega^{n-1}]$$

Base: n = 1, P(1).

$\mathrm{FFT}(P)$

Initialize $n = |P|; \omega = e^{\frac{2\pi i}{n}}$ and $y = [0]^n$
if $n == 1$
    Return P
$P_e = [p_0, p_2, \ldots, p_{n-2}]$ and $P_o = [p_1, p_3, \ldots, p_{n-1}]$
$y_e = \mathrm{FFT}(P_e)$ and $y_o = \mathrm{FFT}(P_o)$
for $j = 0 \cdots n/2 - 1$
    $y[j] = y_e[j] + \omega^j y_o[j]$
    $y[j + n/2] = y_e[j] - \omega^j y_o[j]$
Return y

$P_e(x^2) : [p_0, p_2, \ldots, p_{n-2}]$ $\quad$ $P_o(x^2) : [p_1, p_3, \ldots, p_{n-1}]$

$[\omega^0, \omega^2, \ldots, \omega^{n-2}]$ $\quad$ $[\omega^0, \omega^2, \ldots, \omega^{n-2}]$

$$P(\omega^j) = P_e(\omega^{2j}) + \omega^j P_o(\omega^{2j})$$

$$P(\omega^{j+n/2}) = P_e(\omega^{2j}) - \omega^j P_o(\omega^{2j})$$

$$j \in \{0, 1, \ldots (n/2 - 1)\}$$

Coefficient to Point

Input: $P = [p_0, p_1, \ldots, p_{n-1}]$
Output $y = [P(\omega^0), P(\omega^1), \ldots, P(\omega^{n-1})]$

$\mathrm{FFT}(P)$

```
1  Initialize n = |P|; ω = e^(2πi/n) and y = [0]^n
2  if n == 1
3        Return P
4  P_e = [p_0, p_2, ..., p_{n-2}] and P_o = [p_1, p_3, ..., p_{n-1}]
5  y_e = FFT(P_e) and y_o = FFT(P_o)
6  for j = 0 ... n/2 - 1
7        y[j] = y_e[j] + ω^j y_o[j]
8        y[j + n/2] = y_e[j] - ω^j y_o[j]
9  Return y
```

$P = [4,3,2,1]$
$\mathrm{FFT}(P)$:
1: $n = 4$; $\omega = i$; $y = [0,0,0,0]$.
2-3. 4: $P_e = [4, 2]$ and $P_o = [3, 1]$.
5: $\mathrm{FFT}(P_e)$

Coefficient to Point

Input: $P = [p_0, p_1, \ldots, p_{n-1}]$
Output $y = [P(\omega^0), P(\omega^1), \ldots, P(\omega^{n-1})]$

$\mathrm{FFT}(P)$

1 Initialize $n = |P|; \omega = e^{\frac{2\pi i}{n}}$ and $y = [0]^n$
2 **if** $n == 1$
3 Return P
4 $P_e = [p_0, p_2, \ldots, p_{n-2}]$ and $P_o = [p_1, p_3, \ldots, p_{n-1}]$
5 $y_e = \mathrm{FFT}(P_e)$ and $y_o = \mathrm{FFT}(P_o)$
6 **for** $j = 0 \cdots n/2 - 1$
7 $y[j] = y_e[j] + \omega^j y_o[j]$
8 $y[j + n/2] = y_e[j] - \omega^j y_o[j]$
9 Return y

$P_e = [4,2]$
$\mathrm{FFT}(P_e)$:
1: $n = 2$; $\omega = -1$; $y = [0,0]$.
2-3. 4: $P_e = [4]$ and $P_o = [2]$.
5: $\mathrm{FFT}(P_e) \rightarrow 4$
6: $j = 0$.
7-8: $y[0] = 4 + 2$, $y[1] = 4 - 2$.
9: Return [6, 2].

Coefficient to Point

Input: $P = [p_0, p_1, \ldots, p_{n-1}]$
Output $y = [P(\omega^0), P(\omega^1), \ldots, P(\omega^{n-1})]$

FFT(P)

1 Initialize $n = |P|; \omega = e^{\frac{2\pi i}{n}}$ and $y = [0]^n$
2 **if** $n == 1$
3 Return P
4 $P_e = [p_0, p_2, \ldots, p_{n-2}]$ and $P_o = [p_1, p_3, \ldots, p_{n-1}]$
5 $y_e = \mathrm{FFT}(P_e)$ and $y_o = \mathrm{FFT}(P_o)$
6 **for** $j = 0 \cdots n/2 - 1$
7 $y[j] = y_e[j] + \omega^j y_o[j]$
8 $y[j + n/2] = y_e[j] - \omega^j y_o[j]$
9 Return y

$P = [4,3,2,1]$
FFT(P):
1: $n = 4$; $\omega = i$; $y = [0,0,0,0]$.
2-3. 4: $P_e = [4, 2]$ and $P_o = [3, 1]$.
5: $[6, 2] = \mathrm{FFT}(P_e)$ & $\mathrm{FFT}(P_0)$

Coefficient to Point

Input: P = [p_0, p_1, …, p_{n-1}]
Output y = [P(ω^0), P(ω^1), …, P(ω^{n-1})]

FFT(P)

1 Initialize $n = |P|; \omega = e^{\frac{2\pi i}{n}}$ and $y = [0]^n$
2 **if** $n == 1$
3 Return P
4 $P_e = [p_0, p_2, \ldots, p_{n-2}]$ and $P_o = [p_1, p_3, \ldots, p_{n-1}]$
5 $y_e = \mathrm{FFT}(P_e)$ and $y_o = \mathrm{FFT}(P_o)$
6 **for** $j = 0 \cdots n/2 - 1$
7 $y[j] = y_e[j] + \omega^j y_o[j]$
8 $y[j + n/2] = y_e[j] - \omega^j y_o[j]$
9 Return y

P_0 = [3,1]

FFT(P_o):

1: n = 2; ω = -1; y = [0,0].

2-3. 4: P_e = [3] and P_o = [1].

5: FFT(P_e) → 3 & FFT(P_o) = 1.

6: j = 0

7: y[0] = 3 + 1, 8: y[1] = 3 - 1.

9: Return [4, 2].

Coefficient to Point

$4 + 3x + 2x^2 + x^3$

Input: $P = [p_0, p_1, \ldots, p_{n-1}]$

Output $y = [P(\omega^0), P(\omega^1), \ldots, P(\omega^{n-1})]$

$\mathrm{FFT}(P)$

1 Initialize $n = |P|; \omega = e^{\frac{2\pi i}{n}}$ and $y = [0]^n$

2 **if** $n == 1$

3 Return P

4 $P_e = [p_0, p_2, \ldots, p_{n-2}]$ and $P_o = [p_1, p_3, \ldots, p_{n-1}]$

5 $y_e = \mathrm{FFT}(P_e)$ and $y_o = \mathrm{FFT}(P_o)$

6 **for** $j = 0 \cdots n/2 - 1$

7 $y[j] = y_e[j] + \omega^j y_o[j]$

8 $y[j + n/2] = y_e[j] - \omega^j y_o[j]$

9 Return y

$P = [4,3,2,1]$

FFT(P):

1: n = 4; ω = i; y = [0,0,0,0].

2-3. 4: $P_e = [4, 2]$ and $P_o = [3, 1]$.

5: $[6, 2] = \mathrm{FFT}(P_e)$ & $[4, 2] = \mathrm{FFT}(P_0)$

6: j = 0

7-8: y[0] = 6 + 4, y[2] = 6 - 4.

6: j = 1

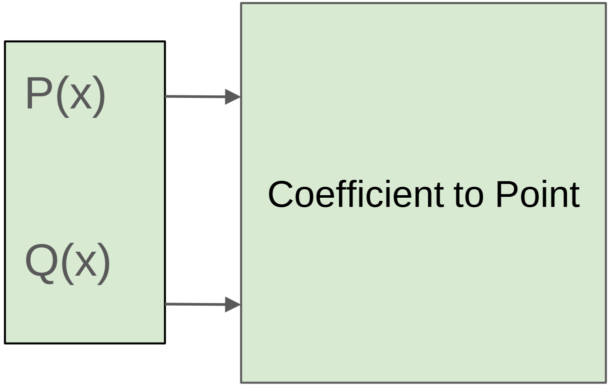
7-8: y[1] = 2 + 2i, y[3] = 2 - 2i

9: [10, 2+2i, 2-2i, 2]

$\omega^0, \omega^1, \omega^2, \omega^3$

Coefficient to Point

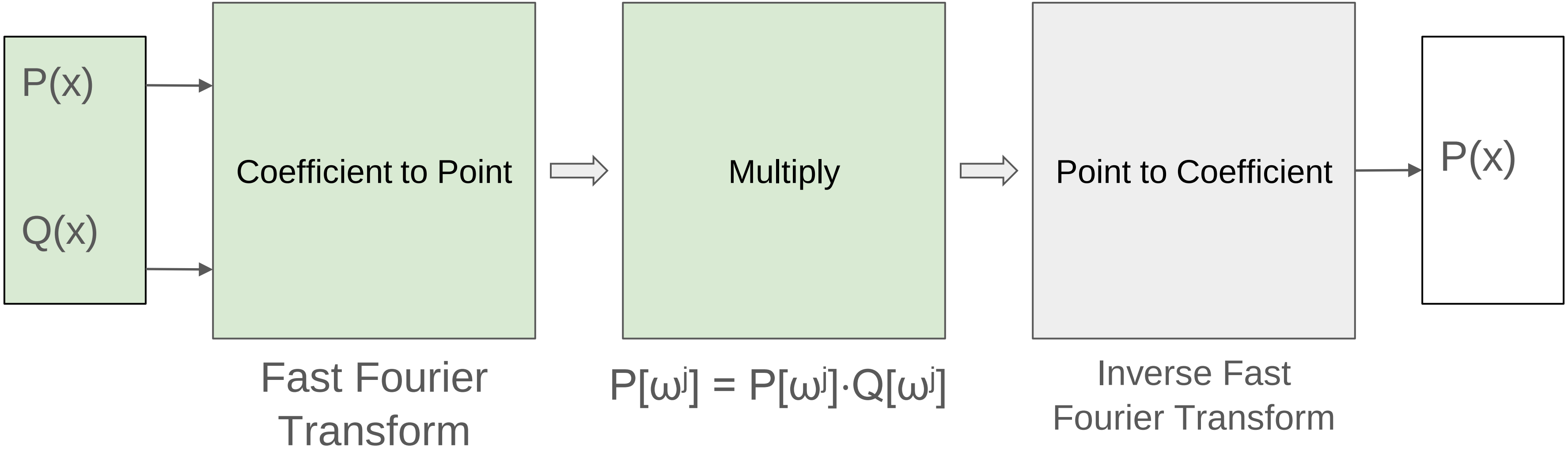
$$\begin{bmatrix} P(\omega^0) \\ P(\omega^1) \\ P(\omega^2) \\ \vdots \\ P(\omega^{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix}$$



Inverse Fast Fourier Transform

Given point value representation of P(x) and Q(x) of degree d. Propose an efficient algorithm to multiply P(x) and Q(x).

$P(x) = P(x) \cdot Q(x)$



Point to Coefficient

Point to Coefficient

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}^{-1}} \begin{bmatrix} P(\omega^0) \\ P(\omega^1) \\ P(\omega^2) \\ \vdots \\ P(\omega^{n-1}) \end{bmatrix}$$

$$\Downarrow$$

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

Replace every ω with 1/(n·ω)

Point to Coefficient

Input: $P = [p_0, p_1, \ldots, p_{n-1}]$

Output $y = [P(\omega^0), P(\omega^1), \ldots, P(\omega^{n-1})]$

IFFT(P)

Initialize $n = |P|; \omega = e^{\frac{-2\pi i}{n}}$ and $y = [0]^n$

if $n == 1$

  Return P

$P_e = [p_0, p_2, \ldots, p_{n-2}]$ and $P_o = [p_1, p_3, \ldots, p_{n-1}]$

$y_e = \text{IFFT}(P_e)$ and $y_o = \text{IFFT}(P_o)$

for $j = 0 \cdots n/2 - 1$

  $y[j] = y_e[j] + \omega^j y_o[j]$

  $y[j + n/2] = y_e[j] - \omega^j y_o[j]$

Return y

$y = y/n.$

FFT(P)

Initialize $n = |P|; \omega = e^{\frac{2\pi i}{n}}$ and $y = [0]^n$

if $n == 1$

  Return P

$P_e = [p_0, p_2, \ldots, p_{n-2}]$ and $P_o = [p_1, p_3, \ldots, p_{n-1}]$

$y_e = \text{FFT}(P_e)$ and $y_o = \text{FFT}(P_o)$

for $j = 0 \cdots n/2 - 1$

  $y[j] = y_e[j] + \omega^j y_o[j]$

  $y[j + n/2] = y_e[j] - \omega^j y_o[j]$

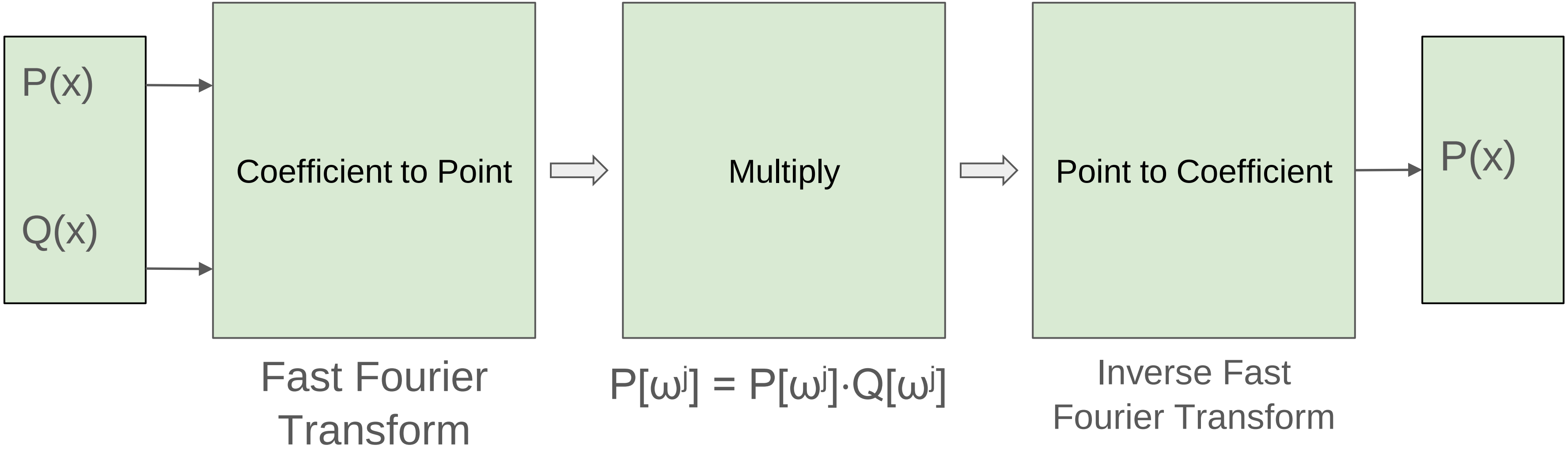
Return y

Observations

- Recurrence: $T(n) = 2T(n/2) + n$
- Running time $O(n \cdot \log(n))$

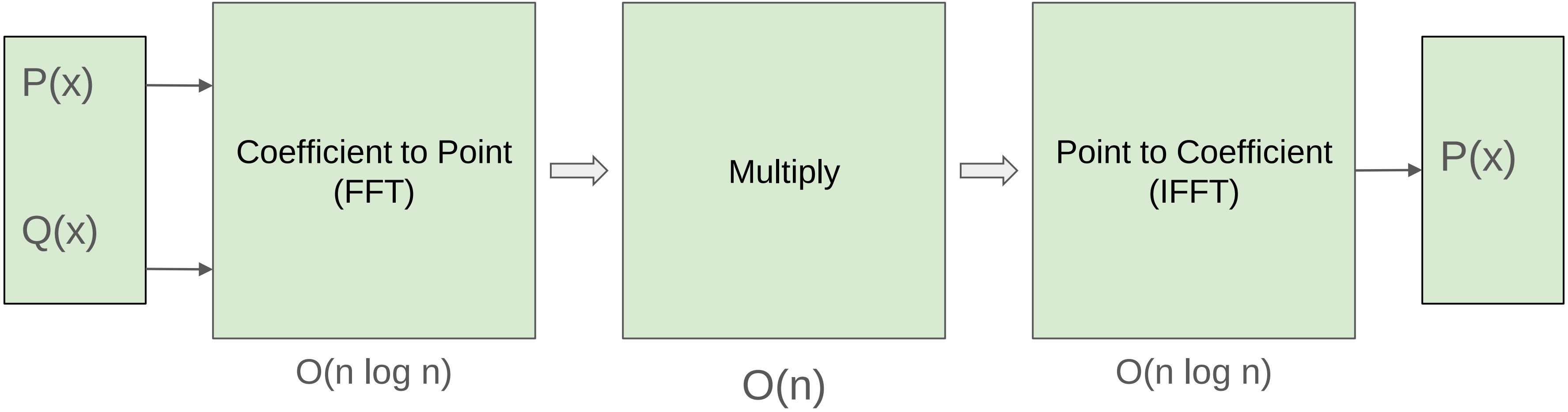
Inverse Fast Fourier Transform

Given point value representation of P(x) and Q(x) of degree d. Propose an efficient algorithm to multiply P(x) and Q(x).



Inverse Fast Fourier Transform

Given point value representation of P(x) and Q(x) of degree d. Propose an efficient algorithm to multiply P(x) and Q(x).



Outline

- Fast Fourier Transform (Contd.)
- Dynamic Programming

Dynamic Programming

Divide & Conquer: Breaks up problem into independent subproblem; solve each subproblem; combine them to get a solution for the original problem.

Dynamic Programming: Breaks up problem into a series of overlapping subproblem; combine solutions of smaller subproblem to to get a solution for the original problem.

Dynamic Programming

The term was coined by *Richard E. Bellman* in 1950s.

Fibonacci Series

Fibonacci series: 0, 1, 1, 2, 3, 5, 8, 13, 21, …

Compute n^{th} fibonacci number recursively.

$F(n) = F(n-1) + F(n-2)$

$F(\cancel{1}\,2) = 1$

$F(\cancel{0}\,1) = 0$

$T(n) = T(n-1) + T(n-2) + 1$

Recurrence: $T(n) = T(n-1) + T(n-2) + 1$

Running time?

Fibonacci Series

Fibonacci series: 0, 1, 1, 2, 3, 5, 8, 13, 21, …

Compute n^{th} fibonacci number recursively.

```
Fib(n)
if M[n] is not empty
        Return M[n]
if n == 0
else
        f = Fib(n − 1) + Fib(n − 2)
M[n] = f
Return f
```

Bottom up!

Reference

Slides

Algorithms Design by Kleinberg & Tardos - 5.6