Complexity & Algorithms - Spring '26

FFTs

Recap

Next topic:

Flows & bipartite matchings

Tuesday → 2 parts

theory & algs

Thursday: applications

Finite Fourier Transform

3 representations for polynomial of deg=n:

1) $p(x) = a_{n-1}x^{n-1} + \cdots a_0$ ← A[0]

Store: coefficients $A[0..n-1]$

→ evaluate?

2) $p(x) = \prod_{i=0}^{n-1} (x - r_i)$ ✓

store: $R[0..n-1]$

3) Samples: $(x_0, \underbrace{p(x_0)}_{y_0}), \ldots, (x_{n-1}, \underbrace{p(x_{n-1})}_{y_{n-1}})$

↳ no

$P(x)$ is unique n-deg poly through these points

Operations

$A(x) \cdot B(x) = C(x)$

Add:
- with coeffs: $c_i = a_i + b_i \Rightarrow O(n)$
- with roots: ??
- with samples: $O(n)$

Multiply:
- with coeffs: $O(n^2)$
- with roots: $O(n)$
- with samples: take $2n$ samples of $O(n)$ each. $A(x_0) \cdot B(x_0) = C(x_0) \rightarrow (x_0, A(x_0) \cdot B(x_0))$

Converting

- Roots are hard!

  Factoring roots is impossible: even for deg = 5, no "easy" formula

- So: coefficients and samples are left.

  Conversion time? $O(n^2)$

Final setup:

representation	evaluate	add	multiply
coefficients	$O(n)$	$O(n)$	$\mathbf{O(n^2)}$
roots + scale	$O(n)$	∞	$O(n)$
samples	$\mathbf{O(n^2)}$	$O(n)$	$O(n)$

$O(n^2)$

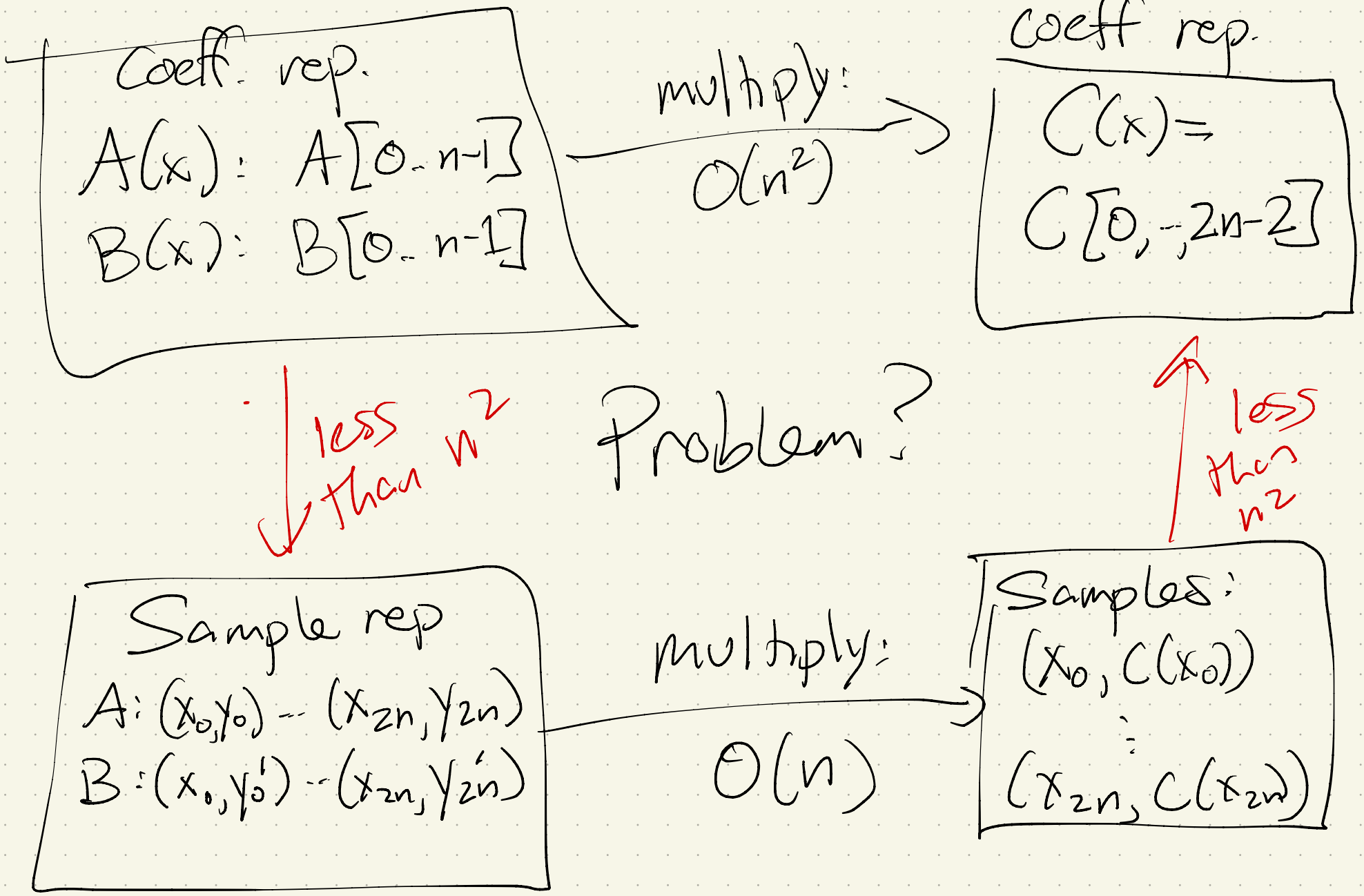
Conversion takes $O(n^2)$ time, via Vandermonde matrix:

samples

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix}.$$

coeffs

Status:


Coeff. rep.
A(x): A[0..n-1]
B(x): B[0..n-1]
multiply:
O(n²)
Coeff rep.
C(x) =
C[0,..,2n-2]
less than n²
Problem?
less than n²
Sample rep
A: (x0,y0) .. (x2n, y2n)
B: (x0,y0') .. (x2n, y2n')
multiply:
O(n)
Samples:
(x0, C(x0))
:
(x2n, C(x2n))

Sadly, conversion takes $O(n^2)$ time.

$$V\vec{a} = \vec{y} \quad \& \quad \vec{a} = V^{-1}\vec{y}$$

(V & V^{-1} take $O(n^3)$ but can precompute if we fix polynomials.)

Goal: Faster!

How? Pick better x_i values, so can divide & conquer.

Divide & Conquer:

$$p(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0$$

Break it up:

$$\deg \le n/2 \quad p_{even}(y) = a_0 + a_2 y + a_4 y^2 + \cdots a_{n-2} y^{\frac{n-2}{2}}$$

$$\deg \le n/2 \quad p_{odd}(y) = a_1 + a_3 y + a_5 y^2 + \cdots a_{n-1} y^{\frac{(n-1)}{2}}$$

Then: $p(x) = p_{even}(x^2) + x \cdot p_{odd}(x^2)$

(squared)

Collapsing set X of size n if

- $|X| = 1$
- or $X^2 = \{x^2 \mid x \in X\}$ is size $n/2$

+ is collapsing

So examples:

- $X = \{1\}$ by def $X^2 = \{x^2 \mid x \in X\} = \{1^2, (-1)^2\} = \{1\}$
- $X = \{1, -1\}$. Why?
- $X = \{1, -1, i, -i\}$. Why?

$X^2 = \{1^2, (-1)^2, i^2, (-i)^2\} = \{1, 1, -1, -1\} = \{1, -1\}$

More generally:

Fix n_0 & some collapsable set X of size n with no $0 \in X$.

Then: $\sqrt{X} = \{\pm\sqrt{x} \mid x \in X\}$ ←

is collapsible & size $2n$.

So: $X_1 = \{1\}$

$X_2 = \{1, -1\}$

$X_4 = \{1, -1, i, -i\}$

$X_8 = \{1, -1, i, -i, \pm\sqrt{i}, \pm\sqrt{-i}\}$

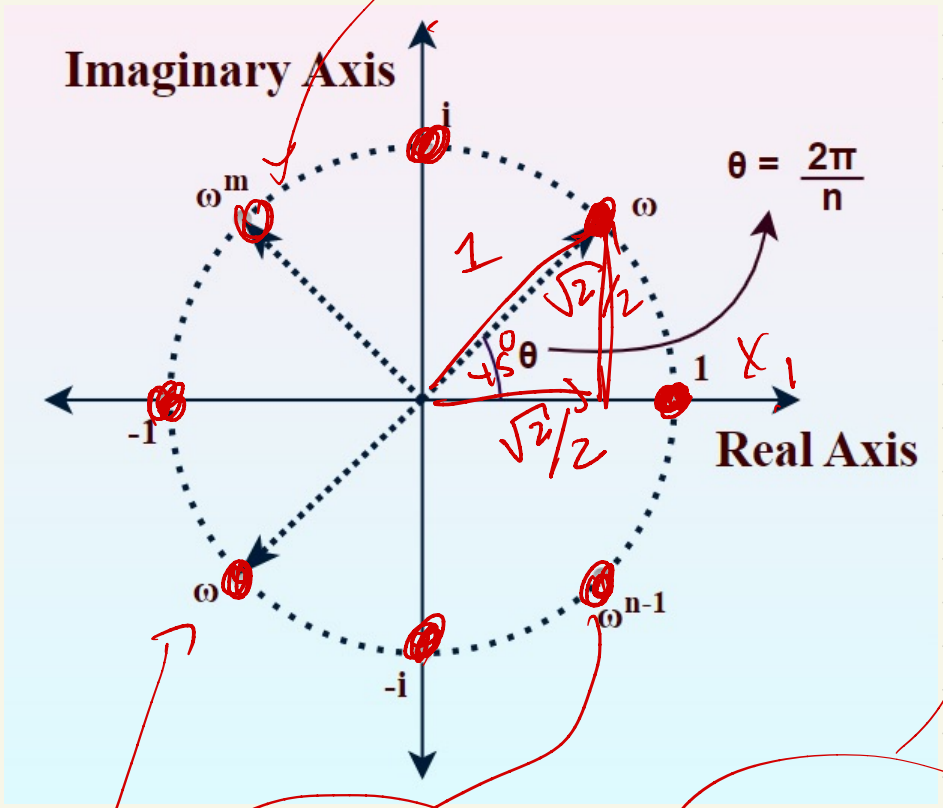
What is X_n?

Roots of unity:

$$\left(\frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}i\right)^2$$

$$\left(\frac{\sqrt{2}}{2}\right)^2 - 2\cdot\frac{\sqrt{2}}{2}\cdot\frac{\sqrt{2}}{2}i + \left(\frac{\sqrt{2}}{2}\right)^2 i^2$$

$$-i$$



$$X_1 = \{1\}$$

$$X_2 = \{1, -1\}$$

$$X_4 = \{1, -1, i, -i\}$$

$$X_8 = \{1, -1, i, -i,$$

$$\frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}i,\ -\frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}i,\ \frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}i,\ -\frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}i\}$$

$$e^{i\pi/4}$$

Why?? Back to $p(x)$:

$$p(x) = p_{even}(x^2) + x \cdot p_{odd}(x^2)$$

($p_{even}(x^2)$: $\deg \le n/2$; $x \cdot p_{odd}(x^2)$: $\deg \le n/2$)

Let X be our samples, s.t. $|X| = n$

A is collapsing. Then:

- we know $X^2 \rightarrow$ size $\le n/2$
- $|\{p_{even}(x^2) \mid x \in X\}| = n/2$
- $|\{p_{odd}(x^2) \mid x \in X\}| = n/2$

Then: For each $x \in X$, compute sample.

$$p(x) = p_{even}(x^2) + x \cdot p_{odd}(x^2)$$

Runtime: $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

↳ pt rep in $O(n \lg n)$

Result: $\{p(x) \mid x \in X\}$

at $|X| = n$

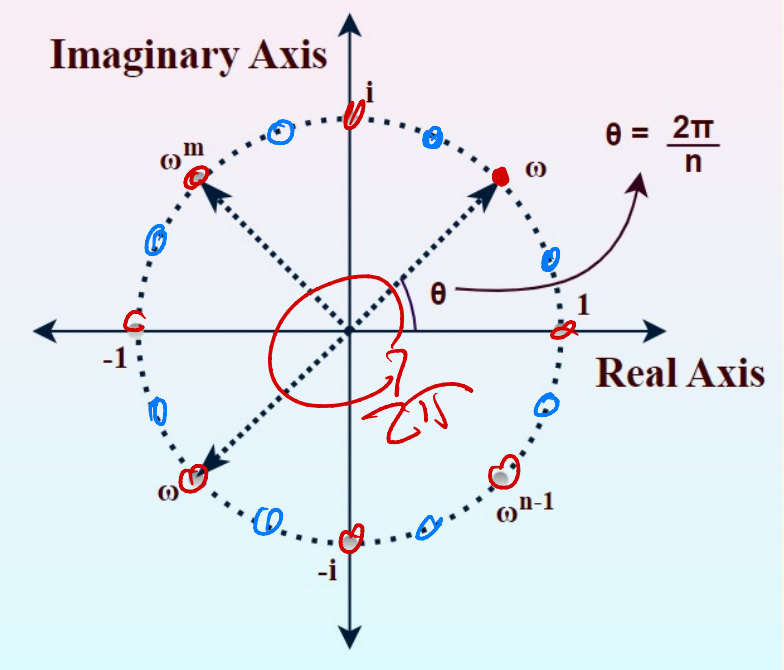
Back to roots of unity!

Too many squared roots!

16th roots

Let $w_n = e^{2\pi i/n}$

$= \cos\left(\frac{2\pi}{n}\right) + i\sin\left(\frac{2\pi}{n}\right)$



Then $w_n^k =$

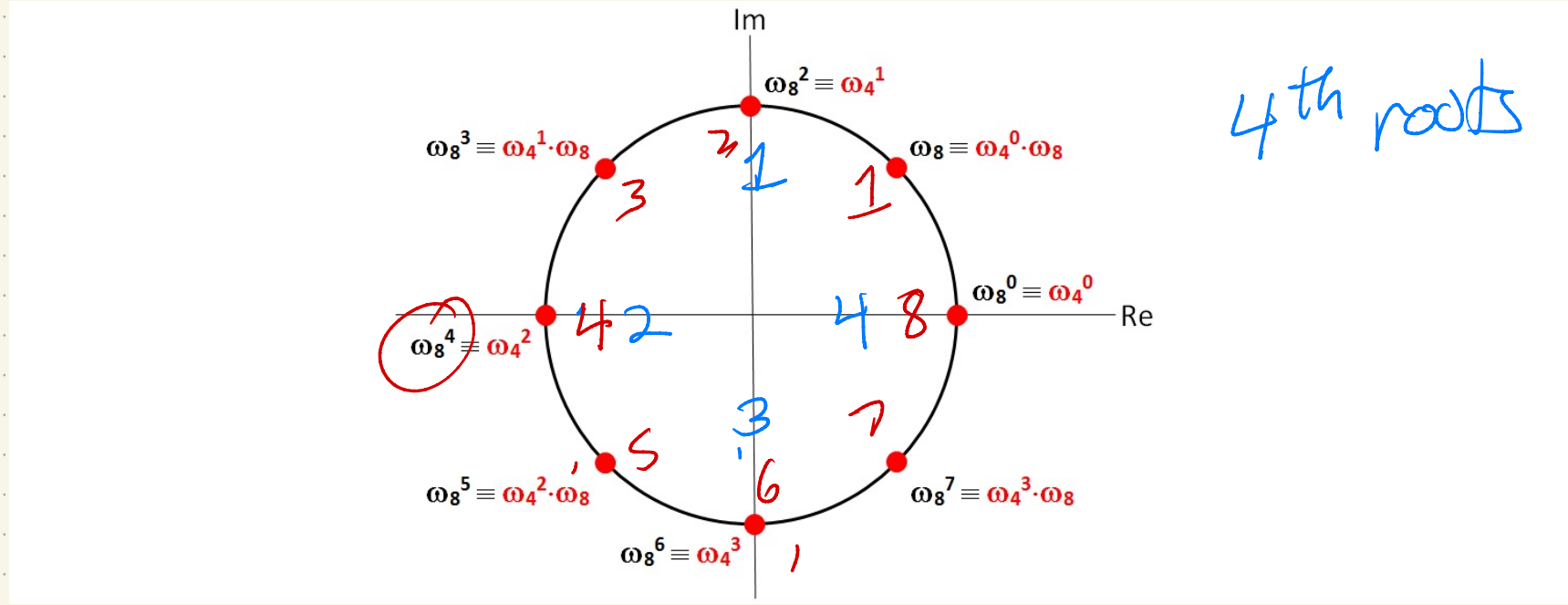
$\cos\left(\frac{2\pi k}{n}\right) + i\sin\left(\frac{2\pi k}{n}\right)$

$= e^{2\pi k/n}$

Why?

They are collapsing!

Power of k = moving k "angles" of ω_n along the circle.



4th roots

So here:

Sample polynomial $P(x)$ at n^{th} roots of unity.

$\Rightarrow$ array $P[0..n-1]$

where $P[i] = p(\omega_n^i)$

$P(1), P(i), P(-1), P(-i)$

$+ P\left(\frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}i\right) \cdots$

Runtime: time $T(n)$ — n = deg of poly (coefficient input)

$T(n) = 2T(\frac{n}{2}) + O(n)$

```
RADIX2FFT(P[0..n-1]):
  if n = 1
      return P

  for j ← 0 to n/2-1
      U[j] ← P[2j]
      V[j] ← P[2j+1]

  U* ← RADIX2FFT(U[0..n/2-1])
  V* ← RADIX2FFT(V[0..n/2-1])

  ω_n ← cos(2π/n) + i sin(2π/n)
  ω ← 1

  for j ← 0 to n/2-1
      P*[j]     ← U*[j] + ω · V*[j]
      P*[j+n/2] ← U*[j] - ω · V*[j]
      ω ← ω · ω_n

  return P*[0..n-1]
```

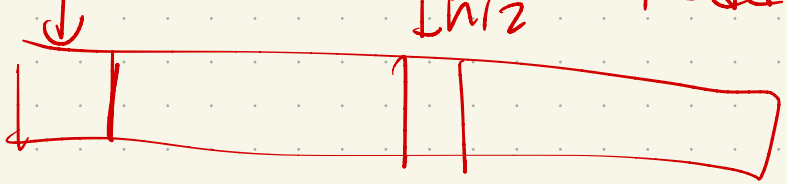
U is Peven

V is Podd

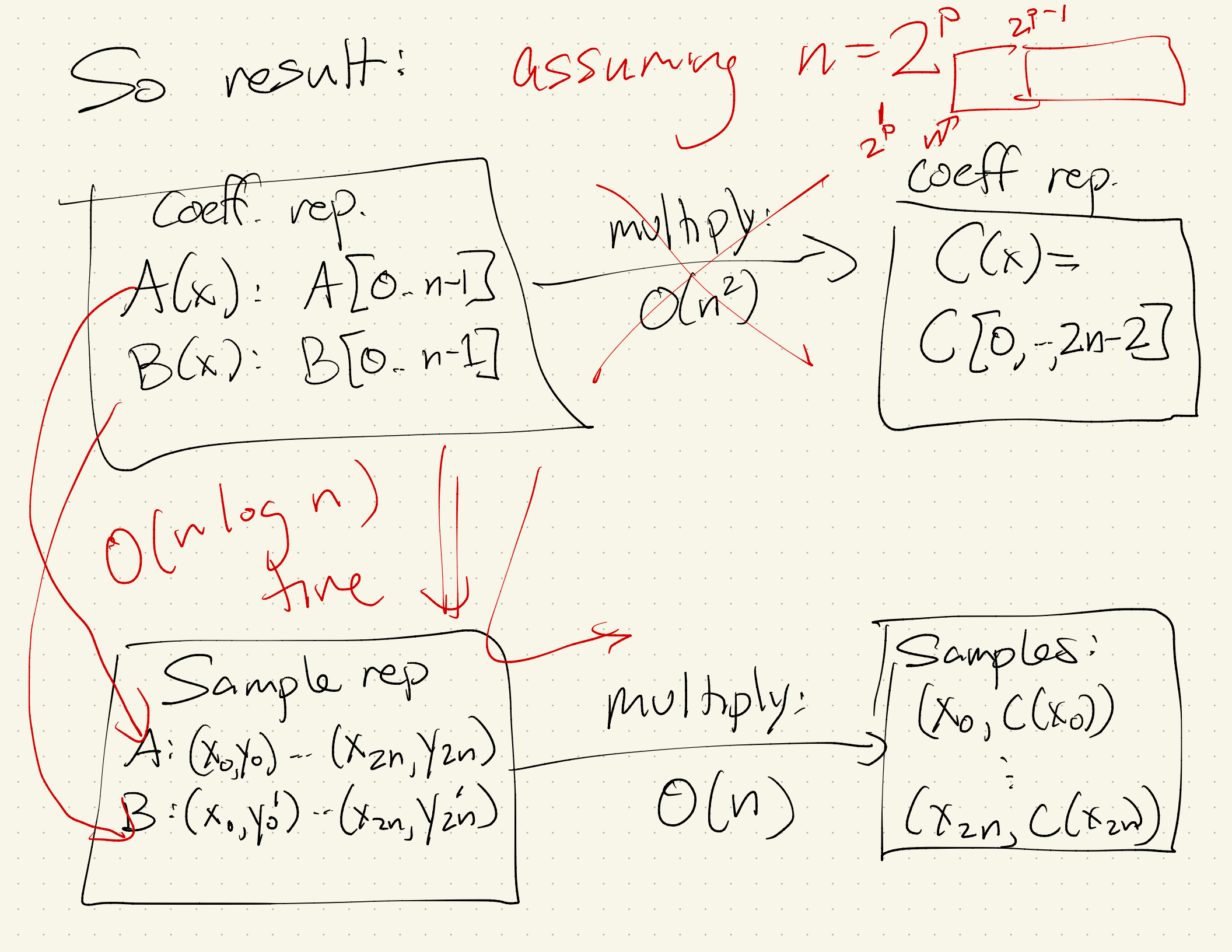
$2 \cdot T(\frac{n}{2})$: U^* is sample pt rep of Peven, V^* sample pt for Podd

sample ⇒

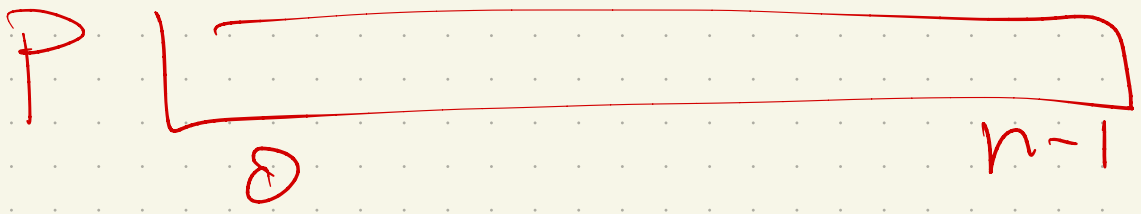
$P(x_i) = P_{even}(x_i^2) + x_i P_{odd}(x_i^2)$



↳ sample point rep is the output

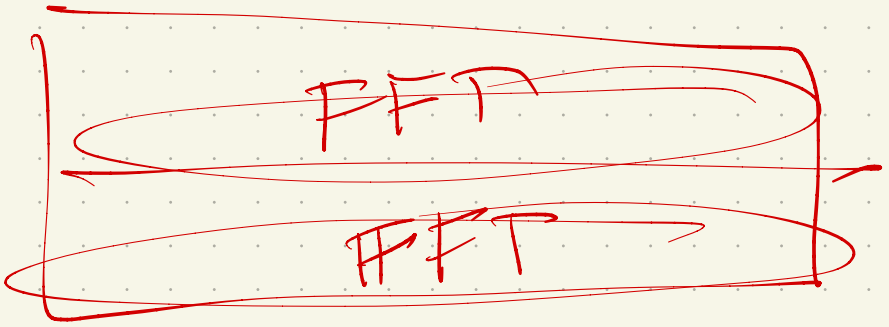

So result:
assuming $n = 2^p$
2^{p-1}
2^p
w
Coeff. rep.
$A(x)$: $A[0..n-1]$
$B(x)$: $B[0..n-1]$
multiply:
$O(n^2)$
Coeff rep.
$C(x) =$
$C[0, \ldots, 2n-2]$
$O(n \log n)$
fine
Sample rep
A: $(x_0, y_0) \ldots (x_{2n}, y_{2n})$
B: $(x_0, y_0') \ldots (x_{2n}, y_{2n}')$
multiply:
$O(n)$
Samples:
$(x_0, C(x_0))$
$(x_{2n}, C(x_{2n}))$

$$n = \frac{n}{2} \cdot \frac{n}{2}$$



↳ Peven

Podd



↳ combined

More generally:

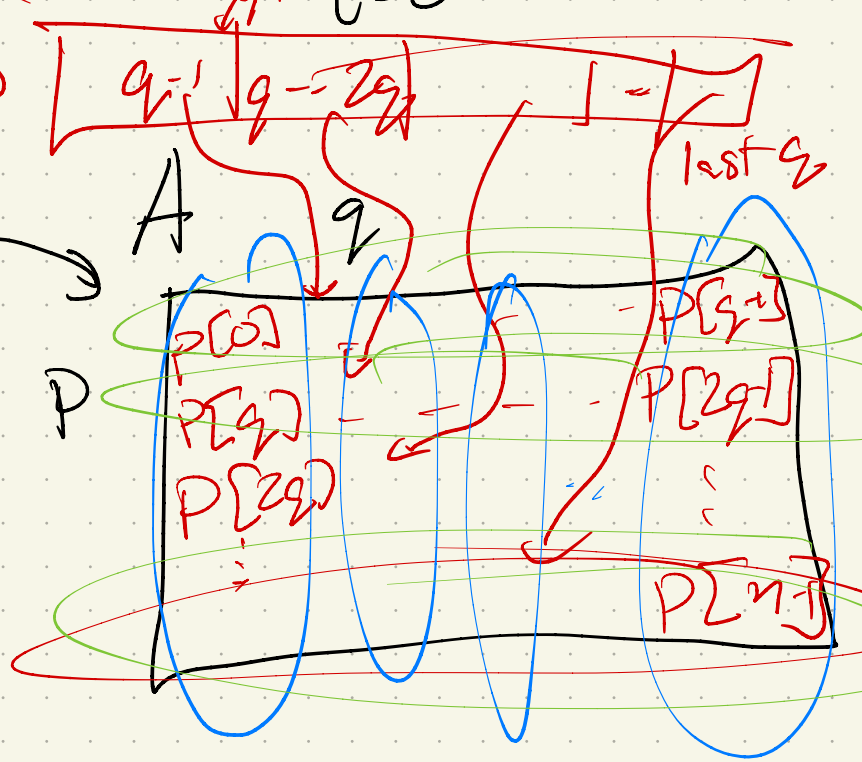
n might not be power of 2

could pad to next higher

So: Let $n = p \cdot q$ & $P(x) = \sum_{i=0}^{n-1} a_i \cdot x^i$

$\deg = \frac{n}{2} \cdot \frac{n}{2}$

Rewrite coeffs: $P[0 \ldots n-1]$



Then: FFT on the columns

& then the rows

```
FACTORFFT(P[0..pq−1]):
  ⟨⟨Copy/typecast to 2d array in row-major order⟩⟩
  for j ← 0 to p−1
      for k ← 0 to q−1
          A[j,k] ← P[jp+k]

  ⟨⟨Recursively apply order-p FFTs to columns⟩⟩
  for k ← 0 to q−1
      B[·,k] ← FFT(A[·,k])

  ⟨⟨Multiply by twiddle factors⟩⟩
  for j ← 0 to p−1
      for k ← 0 to q−1
          B[·,k] ← B[·,k] · ω_pq^jk

  ⟨⟨Recursively apply order-q FFTs to rows⟩⟩
  for j ← 0 to p−1
      C[j,·] ← FFT(C[j,·])

  ⟨⟨Copy/typecast to 1d array in column-major order⟩⟩
  for j ← 0 to p−1
      for k ← 0 to q−1
          P*[j+kq] ← C[j,k]

  return P*[0..pq−1]
```

Figure A.2. The Gauss-Cooley-Tukey FFT algorithm

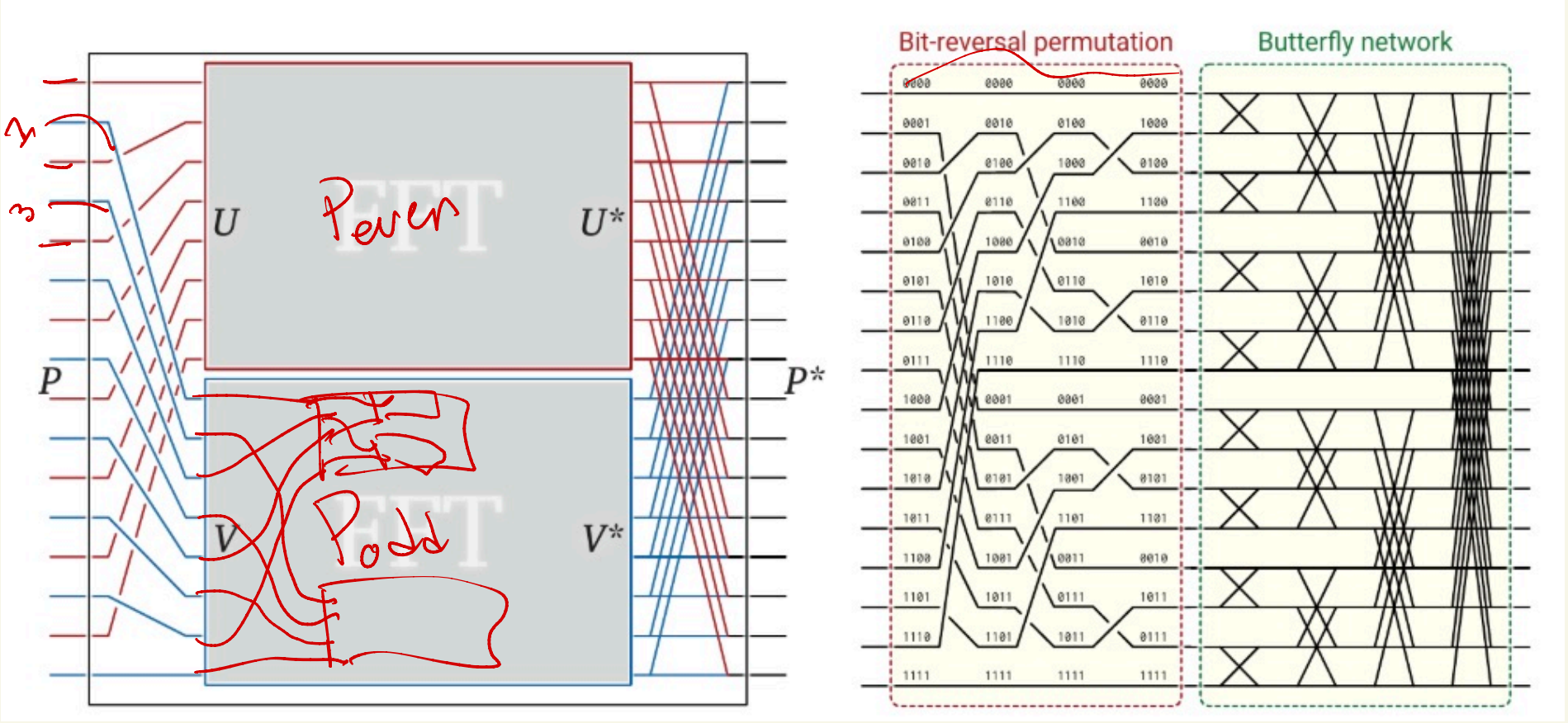
Runtime:

$T(n) = qT(p) + pT(q) + O(n)$

works!

Result: $O(n \log n)$

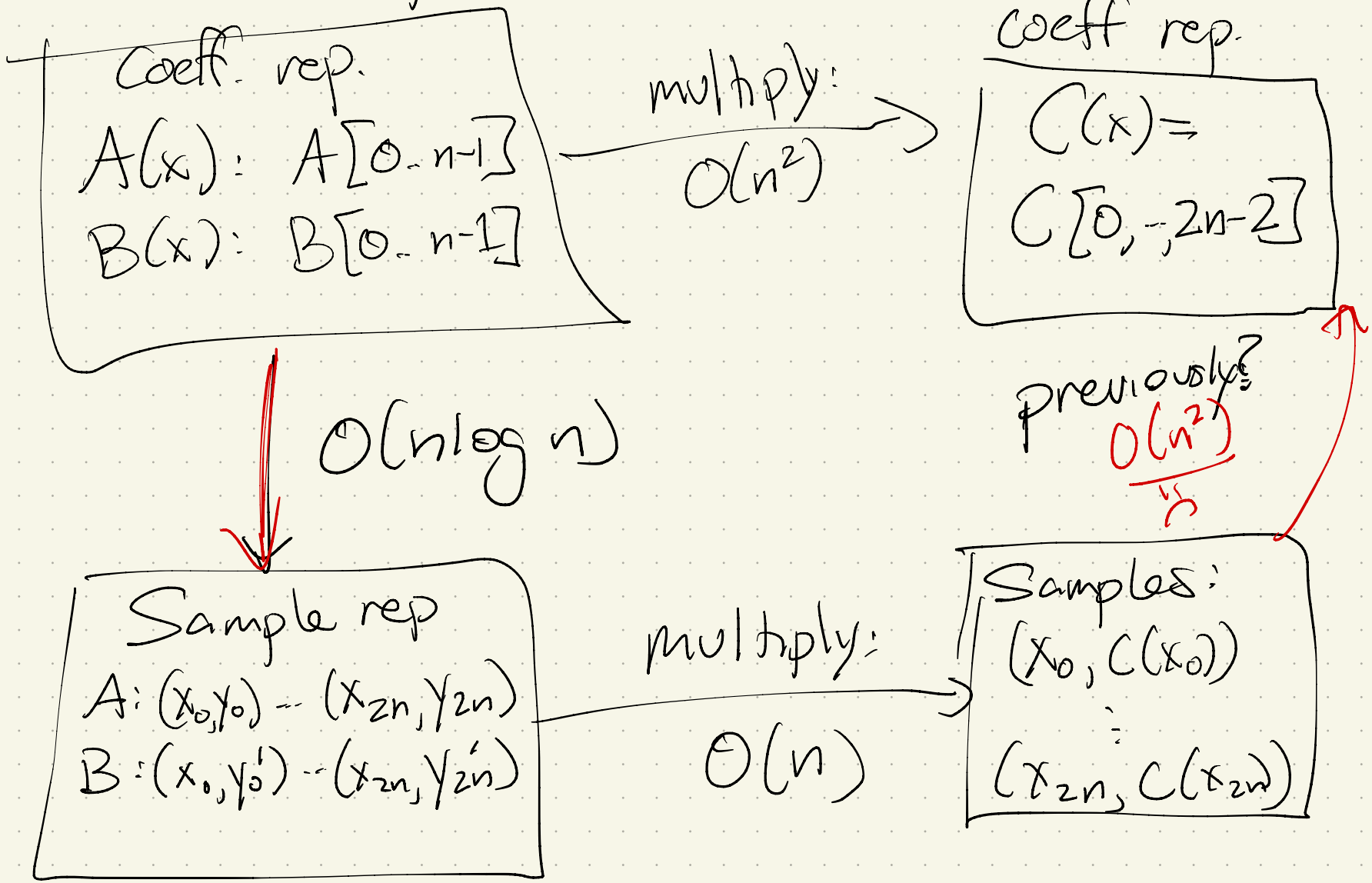
Nice Circuit view:



Unboxing rec, fairy

Not quite done:

Still need to go from samples of product back to coefficients

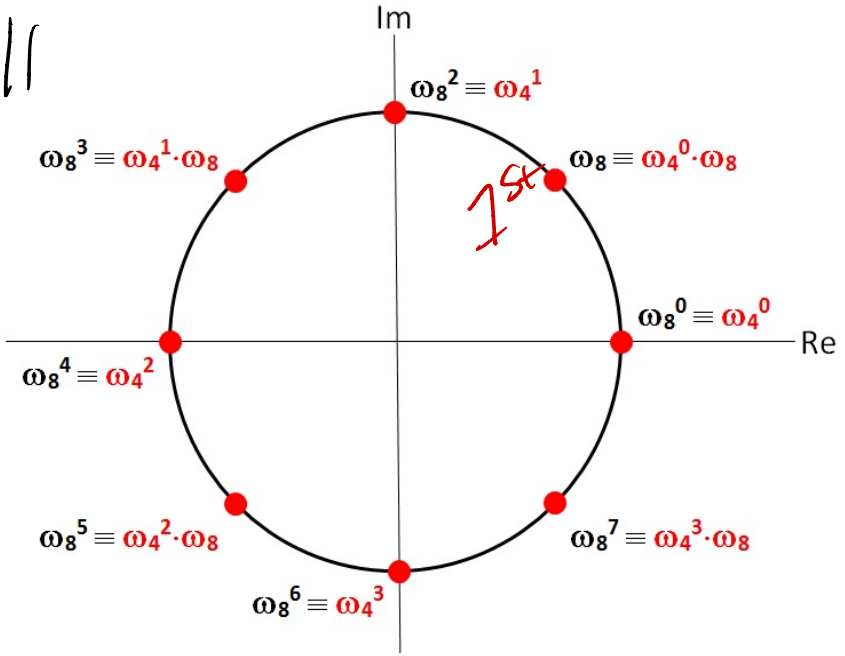


What is our V matrix?

Maybe we can improve things!

$$V = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \omega_n^3 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \omega_n^6 & \cdots & \omega_n^{2(n-1)} \\ 1 & \omega_n^3 & \omega_n^6 & \omega_n^9 & \cdots & \omega_n^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \omega_n^{3(n-1)} & \cdots & \omega_n^{(n-1)^2} \end{bmatrix}$$

These are all roots of unity:



Previously: ← slow

$$\vec{C} = \vec{V}^{-1} \cdot \vec{y}$$

$$V = \begin{bmatrix} & & a+ib & \\ & & & \end{bmatrix}$$

$$V^{-1} = \overline{V} = \begin{bmatrix} & & a-ib/n & \\ & & & \end{bmatrix}$$

Cool fact: If $\vec{V}$ is roots of unity

$q_i = a+ib$

$$\vec{V}^{-1} = \frac{\overline{V}}{n}$$

where the bar is complex conjugation:

$$\overline{a+bi} = a-bi$$

Here: $\overline{\omega_n^k} = \omega_n^{-k} = \frac{1}{\omega_n^k}$

Why does this matter?

Can compute V^{-1}, & reverse the FFT!

$O(n)$

```
INVERSERADIX2FFT(P*[0..n-1]):
  if n = 1
      return P

  for j ← 0 to n/2 - 1
      U*[j] ← P*[2j]
      V*[j] ← P*[2j + 1]

  U ← INVERSERADIX2FFT(U*[0..n/2 - 1])
  V ← INVERSERADIX2FFT(V*[0..n/2 - 1])

  ω_n ← cos(2π/n) - i sin(2π/n)
  ω ← 1

  for j ← 0 to n/2 - 1
      P[j]       ← (U[j] + ω · V[j])/2
      P[j + n/2] ← (U[j] - ω · V[j])/2
      ω ← ω · ω_n

  return P[0..n - 1]
```

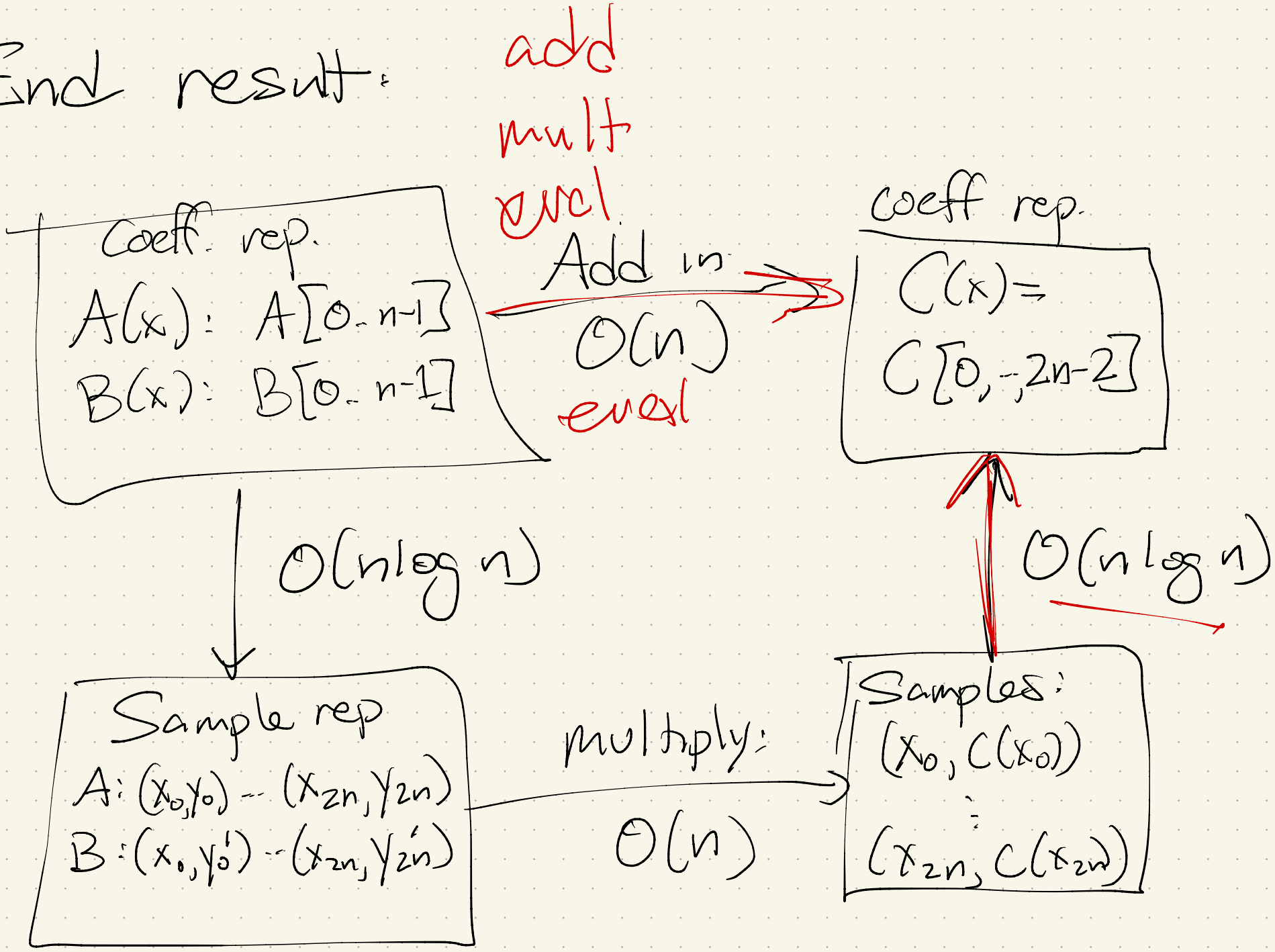
↳ sample rep

↳ coeffs

$T(n) = 2T(n/2) + O(n)$

End result:

add
mult
eval

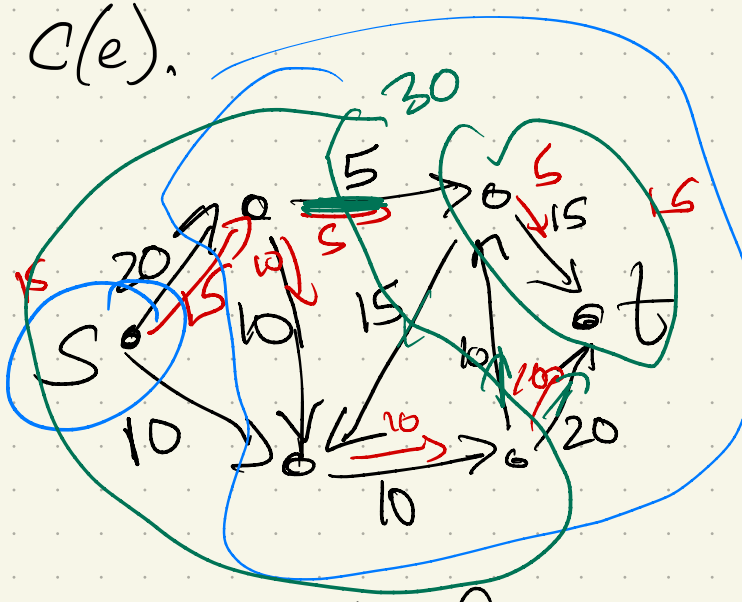


Next ~~time~~: reading

Given a directed graph with two designated vertices, s and t.

Each edge is given a capacity $c(e)$.

Assume: - No edges enter s
- No edges leave t
- Every $c(e) \in \mathbb{Z}^+$



Max flow: Find most I can send from s to t without exceeding edge capacities.

Min cut: find lightest set of edges separating s from t

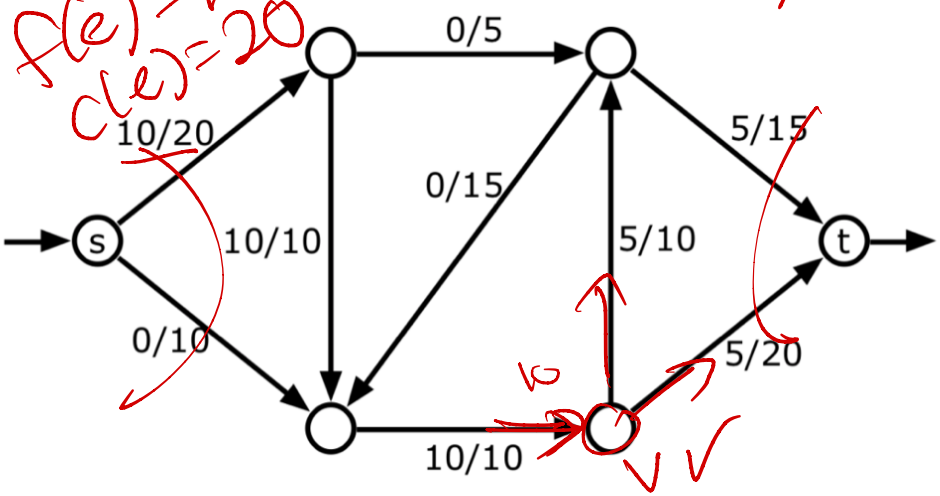
A flow is a function $f: E \to \mathbb{R}^+$, where $f(e)$ is the amount of flow going over edge e

Must satisfy 2 things

Edge constraints: $0 \le f(e) \le c(e)$

Vertex constraints: $\forall v, \sum_{e=u\to v} f(e) = \sum_{e'=v\to x} f(e')$

s, t ✗



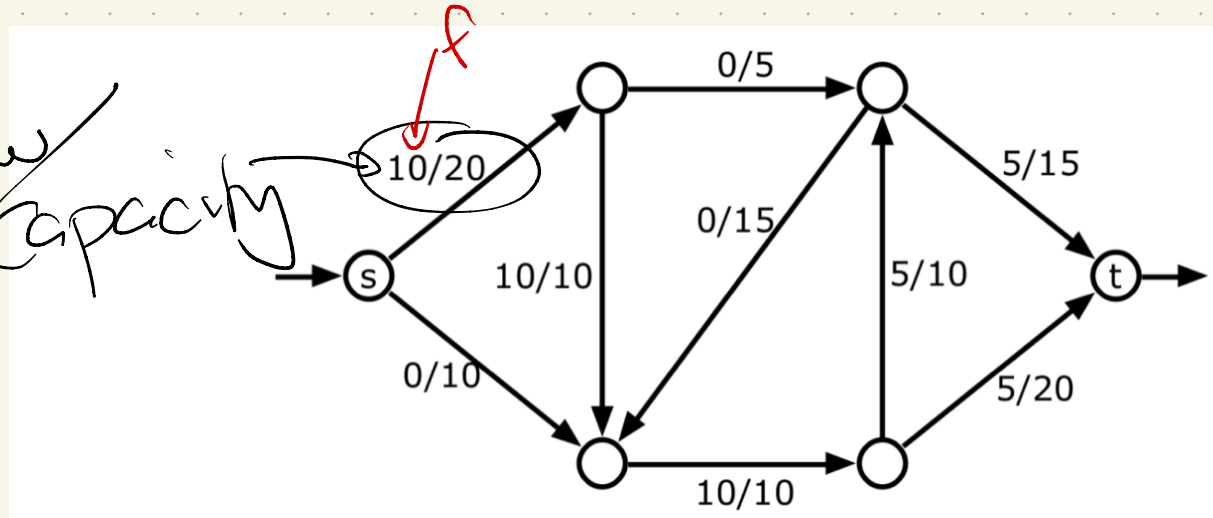
An (s, t)-flow with value 10. Each edge is labeled with its flow/capacity.

Value(f) =

$\sum_{e=s\to v} f(e)$

$= \sum_{e'=u\to t} f(e')$

Note on notation & conventions:



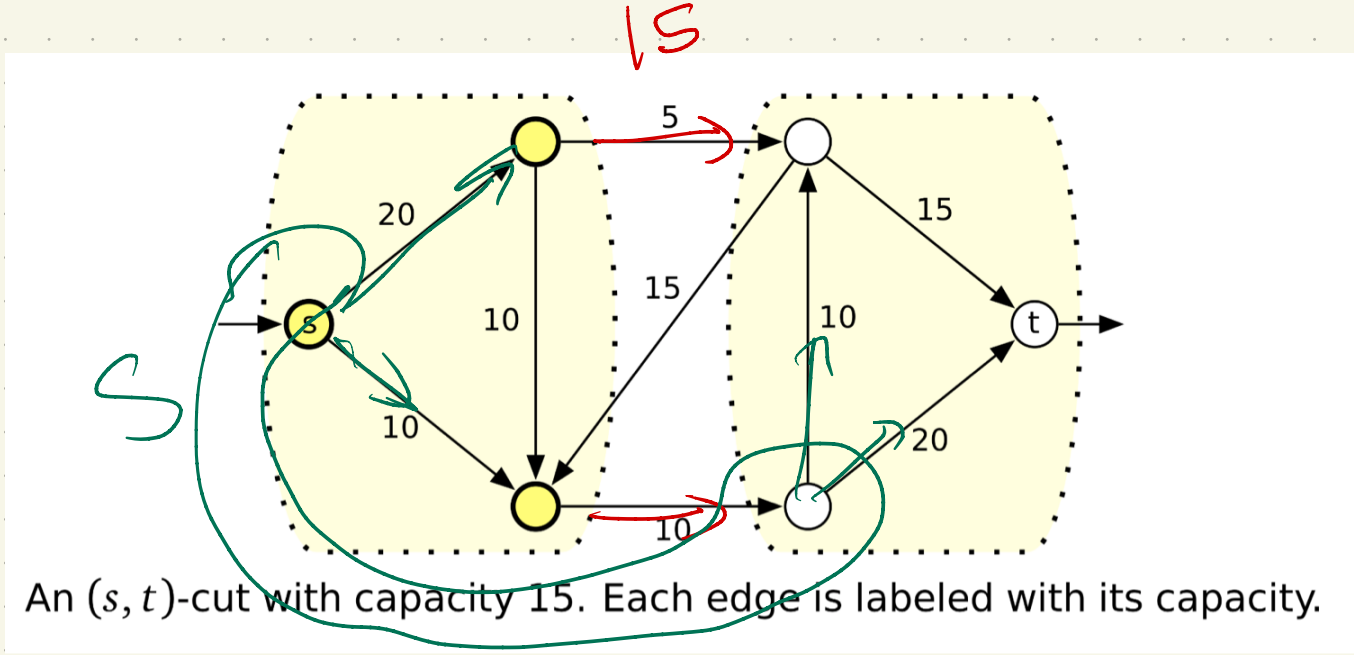
An (s, t)-flow with value 10. Each edge is labeled with its flow/capacity.

A flow is a function on edges!
(so are capacities)
Assume both are positive!
(so vertex constraints make sense)

An s-t cut is a partition of the vertices into 2 sets, S and T, so that

- $s \in S$
- $t \in T$
- $S \cap T = \phi$,

  $S \cup T = V$



An (s, t)-cut with capacity 15. Each edge is labeled with its capacity.

The capacity of a cut is $\sum_{\vec{uv} \in E \text{ with } u \in S, v \in T} c(\vec{uv})$

#cuts: 2^{v-2}

Magic: (duality)

value of
max flow

= capacity of
min cut