Chapter 1
Divide and Conquer

Algorithm Theory
WS 2016/17

Fabian Kuhn

Formulation of the D&C principle

Divide-and-conquer method for solving a problem instance of size n:

1. Divide

$n \leq c$: Solve the problem directly.

$n > c$: Divide the problem into k subproblems of sizes $n_1, \ldots, n_k < n$ $(k \geq \cancel{2})$.

2. Conquer

Solve the k subproblems in the same way (recursively).

3. Combine

Combine the partial solutions to generate a solution for the original instance.

Recurrence Relations: Master Theorem

Recurrence relation

a subproblems · size of subproblems · cost for divide & combine

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n), \qquad T(n) = O(1) \text{ for } n \le n_0$$

$T(n) = 2T(\frac{n}{2}) + O(n^{0.99})$

Cases

- $f(n) = O(n^c),\ c < \log_b a$

  $T(n) = \Theta(n)$

  $$\boldsymbol{T(n) = \Theta\left(n^{\log_b a}\right)}$$

- $f(n) = \Omega(n^c),\ c > \log_b a$

  $T(n) = 2T(\frac{n}{2}) + \Theta(n^{1.01})$

  $T(n) = \Theta(n^{1.01})$

  $$\boldsymbol{T(n) = \Theta\big(f(n)\big)}$$

- $f(n) = \Theta\left(n^c \cdot \log^k n\right),\ c = \log_b a$

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

  $k \geq 0$

  $$\boldsymbol{T(n) = \Theta\left(n^c \cdot \log^{k+1} n\right)}$$

  $\Theta(n \log n)$

  $f(n) \cdot \log(n)$

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

  $\Theta(n \log\log n)$

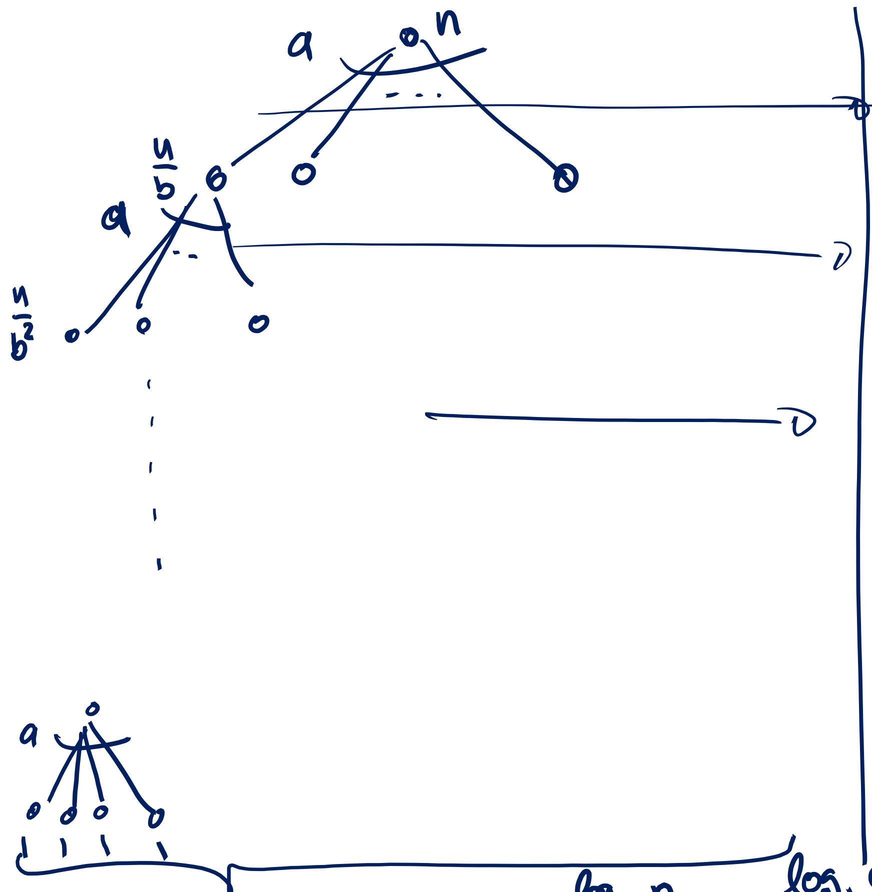
Recurrence Relations: Master Theorem

Recurrence relation

O

$f(n) = n^c$

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n), \qquad T(n) = O(1) \ \text{for } n \le n_0$$



$f(n) = n^c$

$a \cdot f\left(\frac{n}{b}\right) = \frac{a}{b^c} n^c$

$a^2 f\left(\frac{n}{b^2}\right) = \left(\frac{a}{b^c}\right)^2 \cdot n^c$

$\vdots$

$\frac{a}{b^c} < 1 \rightarrow T(n) = f(n)$

$b^c > a$

$c > \log_b a$

$a^{\log_b n} = n^{\log_b a}$

$a^{\log_b n} = n^{\log_b a}$

Polynomials

Real polynomial p in one variable x:

$a_i \in \mathbb{R}$

$$p(x) = \overbrace{a_{n-1}x^{n-1}}^{\text{degree } = n-1} + \dots + a_1 x^1 + a_0$$

Coefficients of p: $a_0, a_1, \dots, a_n \in \mathbb{R}$

Degree of p: largest power of x in p ($n - 1$ in the above case)

Example:

$$p(x) = 3x^3 - 15x^2 + 18x + 0$$

$$a = (a_0, a_1, a_2, a_3) = (0, 18, -15, 3)$$

Set of all real-valued polynomials in x: $\mathbb{R}[x]$ (polynomial ring)

Operations: Evaluation

- Given: Polynomial $p \in \mathbb{R}[x]$ of degree $n-1$

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

Operations: Evaluation

assumption
basic op. on reals cost $O(1)$

- Given: Polynomial $p \in \mathbb{R}[x]$ of degree $n-1$

$$p(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x + a_0$$

- **Horner's method** for evaluation at specific value x_0:

$$p(x_0) = \left(\ldots\left(\left(a_{n-1}x_0 + a_{n-2}\right)x_0 + a_{n-3}\right)x_0 + \cdots + a_1\right)x_0 + a_0$$

- Pseudo-code:

$p \coloneqq a_{n-1};\ i \coloneqq n-1;$
while $(i > 0)$ **do**
 $i \coloneqq i - 1;$
 $p \coloneqq p \cdot x_0 + a_i$

- Running time: $\boxed{O(n)}$

Operations: Addition

- Given: Polynomials $p, q \in \mathbb{R}[x]$ of degree $n-1$

$$p(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0$$
$$q(x) = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \cdots + b_1 x + b_0$$

- Compute sum $p(x) + q(x)$:

$c_i = a_i + b_i$

$$\begin{aligned} p(x) + q(x) &= (a_{n-1}x^{n-1} + \cdots + a_0) + (b_{n-1}x^{n-1} + \cdots + b_0) \\ &= (a_{n-1} + b_{n-1})x^{n-1} + \cdots + (a_1 + b_1)x + (a_0 + b_0) \end{aligned}$$

time: $O(n)$

Operations: Multiplication

- Given: Polynomials $p, q \in \mathbb{R}[x]$ of degree $n-1$

$$p(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$
$$q(x) = b_{n-1}x^{n-1} + \cdots + b_1 x + b_0$$

$c_0 = a_0 b_0$, $c_1 = a_0 b_1 + a_1 b_0$, $c_2 = a_0 b_2 + a_1 b_1 + a_2 b_0$

- Product $p(x) \cdot q(x)$:

$$p(x) \cdot q(x) = (a_{n-1}x^{n-1} + \cdots + a_0) \cdot (b_{n-1}x^{n-1} + \cdots + b_0)$$
$$= c_{2n-2}x^{2n-2} + c_{2n-3}x^{2n-3} + \cdots + c_1 x + c_0$$

- Obtaining c_i: what products of monomials have degree i?

$$\text{For } 0 \leq i \leq 2n-2\text{: } c_i = \sum_{j=0}^{i} a_j b_{i-j}$$

where $a_i = b_i = 0$ for $i \geq n$.

- Running time naïve algorithm: $O(n^2)$

Faster Multiplication?

- Multiplication is slow $\left(\Theta(n^2)\right)$ when using the (standard coefficient representation)
- Try divide-and-conquer to get a faster algorithm
- Assume: degree is $n-1$, n is even   n is power of 2
- Divide polynomial $p(x) = a_{n-1}x^{n-1} + \cdots + a_0$ into 2 polynomials of degree $n/_2 - 1$:

$n-1$

$n/_2-1$

$$p_0(x) = a_{n/_2-1}x^{n/_2-1} + \cdots + a_0$$
$$p_1(x) = \left(a_{n-1}x^{n/_2-1} + \cdots + a_{n/_2}\right) \times x^{n/2}$$
$$p(x) = p_1(x) \cdot x^{n/_2} + p_0(x)$$

- Similarly: $q(x) = q_1(x) \cdot x^{n/_2} + q_0(x)$

Use Divide-And-Conquer

- **Divide:**

  $p(x) = p_1(x) \cdot x^{n/2} + p_0(x), \qquad q(x) = q_1(x) \cdot x^{n/2} + q_0(x)$

- **Multiplication:**

  $$p(x)q(x) = p_1(x)q_1(x) \cdot x^n + (p_0(x)q_1(x) + p_1(x)q_0(x)) \cdot x^{n/2} + p_0(x)q_0(x)$$

- 4 multiplications of degree $n/2 - 1$ polynomials:

  $$T(n) = 4T(n/2) + O(n)$$

  $\log_b a = \log_2 4 = 2$

- Leads to $T(n) = \Theta(n^2)$ like the naive algorithm…
  - follows immediately by using the master theorem

More Clever Recursive Solution

- **Recall that**

$$\underline{p(x)q(x)} = \overbrace{p_1(x)q_1(x)}^{A} \cdot x^n + \underbrace{\left(p_0(x)q_1(x) + p_1(x)q_0(x)\right)}_{B} \cdot x^{n/2} + \underbrace{p_0(x)q_0(x)}_{C}$$

- Compute $r(x) = \left(\underline{p_0(x) + p_1(x)}\right) \cdot \left(\underline{q_0(x) + q_1(x)}\right)$:

$$r(x) = \underbrace{p_0(x)q_0(x)}_{C} + \underbrace{p_0(x)q_1(x) + p_1(x)q_0(x)}_{B} + \underbrace{p_1(x)q_1(x)}_{A} = A + B + C$$

<u>compute:</u> <u>$r(x)$, A, C</u>

$$B = r(x) - A - C$$

Karatsuba Algorithm

- Recursive multiplication:

$$r(x) = \big(p_0(x) + p_1(x)\big) \cdot \big(q_0(x) + q_1(x)\big)$$

$$\begin{aligned} p(x)q(x) = {} & p_1(x)q_1(x) \cdot x^n \\ & + \big(r(x) - p_0(x)q_0(x) + p_1(x)q_1(x)\big) \cdot x^{n/2} \\ & + p_0(x)q_0(x) \end{aligned}$$

- Recursively do 3 multiplications of degr. $(n/2 - 1)$-polynomials

$$T(n) = 3T(n/2) + O(n)$$

$\log_2 3$

- Gives: $T(n) = O(n^{1.58496\ldots})$ (see Master theorem)

Representation of Polynomials

Coefficient representation:

- Polynomial $p(x) \in \mathbb{R}[x]$ of degree $n - 1$ is given by its n coefficients $a_0, \dots, a_{n-1}$:

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

- Coefficient vector $\boldsymbol{a} = (a_0, a_1, \dots a_{n-1})$

- Example:

$$p(x) = 3x^3 - 15x^2 + 18x$$

- The most typical (and probably most natural) representation of polynomials

Representation of Polynomials

Product of linear factors:

- Polynomial $p(x) \in \mathbb{C}[x]$ of degr. $n-1$ is given by its $n-1$ roots

$$p(x) = a_{n-1} \cdot (x - x_1) \cdot (x - x_2) \cdot \ldots \cdot (x - x_{n-1})$$

- Example:

$$p(x) = 3x(x-2)(x-3)$$

roots : 0, 2, 3

- Every polynomial has exactly $n-1$ roots $x_i \in \mathbb{C}$ (s.t. $p(x_i) = 0$)
  - Polynomial is uniquely defined by the $n-1$ roots and a_{n-1}

- We will not use this representation…

Representation of Polynomials

Point-value representation:

- Polynomial $p(x) \in \mathbb{R}[x]$ of degree $n-1$ is given by n point-value pairs:

$$p = \{(x_0, p(x_0)), (x_1, p(x_1)), \dots, (x_{n-1}, p(x_{n-1}))\}$$

where $x_i \neq x_j$ for $i \neq j$.

- Example: The polynomial

$$p(x) = 3x(x-2)(x-3)$$

is uniquely defined by the four point-value pairs $(0,0), (1,6), (2,0), (3,0)$.

Operations: Coefficient Representation

$$\boldsymbol{p(x) = a_{n-1}x^{n-1} + \cdots + a_0, \qquad q(x) = b_{n-1}x^{n-1} + \cdots + b_0}$$

Evaluation: Horner's method: Time $O(n)$

Addition:

$$p(x) + q(x) = (a_{n-1} + b_{n-1})x^{n-1} + \cdots + (a_0 + b_0)$$

- Time: $O(n)$

Multiplication:

$$p(x) \cdot q(x) = c_{2n-2}x^{2n-2} + \cdots + c_0, \qquad \text{where } c_i = \sum_{j=0}^{i} a_j b_{i-j}$$

- Naive solution: Need to compute product $a_i b_j$ for all $0 \leq i, j \leq n$

1.58

- Time: $O(n^{\cancel{2}})$

Operations: Linear Factors (Roots)

$$\boldsymbol{p(x) = a_{n-1} \cdot (x - x_1) \cdot \ldots \cdot (x - x_{n-1})}$$
$$\boldsymbol{q(x) = b_{n-1} \cdot (x - x_1) \cdot \ldots \cdot (x - x_{n-1})}$$

Evaluation:

- Just plug in the value where the poly. is evaluated: Time $O(n)$

Multiplication:

- Concatenate the two representations: Time $O(n)$

Addition:

- Need to find the roots of $p(x) + q(x)$
- For polynomials of degree ≥ 4, this is not possible by using basic arithmetic operations ($+, -, \cdot, /, \sqrt[a]{b}$)
- In the usual computational model impossible
  - Numerically, the roots can be computed to arbitrary precision

Operations: Point-Value Representation

$$\boldsymbol{p = \{(x_0, p(x_0)), \dots, (x_{n-1}, p(x_{n-1}))\}}$$
$$\boldsymbol{q = \{(x_0, q(x_0)), \dots, (x_{n-1}, q(x_{n-1}))\}}$$

- Note: we use the **same points $\boldsymbol{x_0, \dots, x_{n-1}}$** for both polynomials

Addition:

$$p + q = \{(x_0, p(x_0) + q(x_0)), \dots, (x_{n-1}, p(x_{n-1}) + q(x_{n-1}))\}$$

- Time: $O(n)$

Multiplication:

$$p \cdot q = \{(x_0, p(x_0) \cdot q(x_0)), \dots, (x_{2n-2}, p(x_{2n-2}) \cdot q(x_{2n-2}))\}$$

- Time: $O(n)$ need $2n-1$ points

Evaluation: Polynomial interpolation can be done in $O(n^2)$

Operations on Polynomials

Cost depending on representation:

	Coefficient	~~Roots~~	Point-Value
Evaluation	$O(n)$	$O(n)$	$O(n^2)$
Addition	$O(n)$	∞	$O(n)$
Multiplication	$O(n^{1.58})$	$O(n)$	$O(n)$

$O(n \log n)$

Cost??

Faster Polynomial Multiplication?

Multiplication is fast when using the point-value representation

Idea to compute $p(x) \cdot q(x)$ (for polynomials of degree $< n$):

p, q of degree $n-1$, n coefficients

↓ **Evaluation** at points $x_0, x_1, \dots, x_{2n-2}$ $N-1$ naively: $O(n^2)$

$2 \times 2n$ point-value pairs $\big(x_i, p(x_i)\big)$ and $\big(x_i, q(x_i)\big)$

↓ **Point-wise multiplication**

$2n$ point-value pairs $\big(x_i, p(x_i)q(x_i)\big)$

↓ **Interpolation**

$p(x)q(x)$ of degree $2n-2$, $2n-1$ coefficients

Coefficients to Point-Value Representation

Given: Polynomial $p(x)$ by the coefficient vector $(a_0, a_1, \dots, a_{N-1})$

Goal: Compute $p(x)$ for all x in a given set X $= (x_0, \dots, x_{N-1})$

- Where X is of size $|X| = N$
- Assume that N is a power of 2

$\boxed{\text{compute } p(x)\ \forall x \in X}$

Divide and Conquer Approach

- Divide $p(x)$ of degree $N-1$ (N is even) into 2 polynomials of degree ${}^{N}/_{2} - 1$ differently than in Karatsuba's algorithm
- $p_0(y) = a_0 + a_2 y + a_4 y^2 + \cdots + a_{N-2} y^{N/2-1}$ (even coeff.)
  $p_1(y) = a_1 + a_3 y + a_5 y^2 + \cdots + a_{N-1} y^{N/2-1}$ (odd coeff.)

$\rightarrow (a_0, a_2, a_4, \dots, a_{N-2})$

$\rightarrow (a_1, a_3, \dots, a_{N-1})$

Coefficients to Point-Value Representation

Goal: Compute $p(x)$ for all x in a given set X of size $|X| = N$

- Divide $p(x)$ of degr. $N-1$ into 2 polynomials of degr. $N/2 - 1$

$$p_0(y) = a_0 + a_2 y + a_4 y^2 + \cdots + a_{N-2} y^{N/2-1} \quad \text{(even coeff.)}$$
$$p_1(y) = a_1 + a_3 y + a_5 y^2 + \cdots + a_{N-1} y^{N/2-1} \quad \text{(odd coeff.)}$$

Let's first look at the "combine" step:

- We need to compute $p(x)$ for all $x \in X$ after recursive calls for polynomials p_0 and p_1:

$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots a_4 x^4$

$\forall x \in X: \; p(x) = P_0(x^2) + \underbrace{x \cdot P_1(x^2)}_{x(a_1 + a_3 x^2 + a_5 x^4 + \ldots)}$

cost of combine

$O(|X|)$

recursively compute $P_0(y)$, $P_1(y)$ for all $y \in X^2$

$X^2 := \{x^2 : x \in X\}$

Coefficients to Point-Value Representation

Goal: Compute $p(x)$ for all x in a given set X of size $|X| = N$

- Divide $p(x)$ of degr. $N-1$ into 2 polynomials of degr. $N/2 - 1$

$$p_0(y) = a_0 + a_2 y + a_4 y^2 + \cdots + a_{N-2} y^{N/2-1} \quad \text{(even coeff.)}$$
$$p_1(y) = a_1 + a_3 y + a_5 y^2 + \cdots + a_{N-1} y^{N/2-1} \quad \text{(odd coeff.)}$$

Let's first look at the "combine" step:

$$\forall x \in X: \quad p(x) = p_0(x^2) + x \cdot p_1(x^2)$$

- Recursively compute $\underline{p_0(y)}$ and $\underline{p_1(y)}$ for all $\underline{y \in X^2}$
  - Where $X^2 := \{x^2 : x \in X\}$

- Generally, we have $|X^2| = |X|$

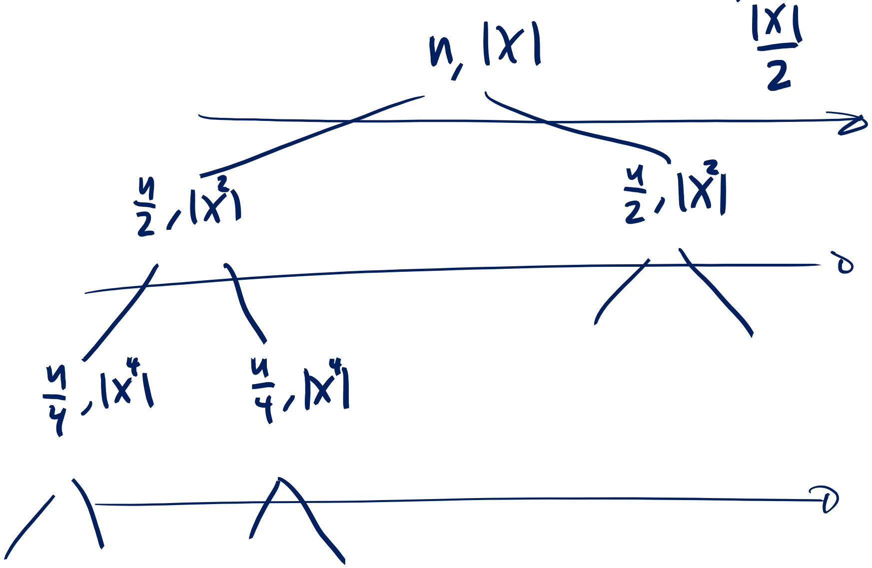
Analysis

$T(n,n) = ?$ $\qquad T(n) = 2T(n/2) + O(n)$

Recurrence formula for the given algorithm:

$$T(n, |X|) = 2 \cdot T\left(\frac{n}{2}, |X|\right) + O(n + |X|)$$

$\frac{|X|}{2}$



$n + |X|$

$2\left(\frac{n}{2} + |X|\right) = n + 2|X|$

$4\left(\frac{n}{4} + |X|\right) = n + 4|X|$

$\vdots$

$n(1 + |X|) = n + n|X|$

$|X| = n$

$O(n^2)$

$|X^2| \overset{?}{=} |X|$

Faster Algorithm?

- In order to have a faster algorithm, we need $|X^2| < |X|$

$$\text{if } |X^2| = \frac{|X|}{2} \Longrightarrow O(n \log_2 n) \text{ alg.}$$

$n \to \frac{n}{2} \to \frac{n}{4} \ldots$

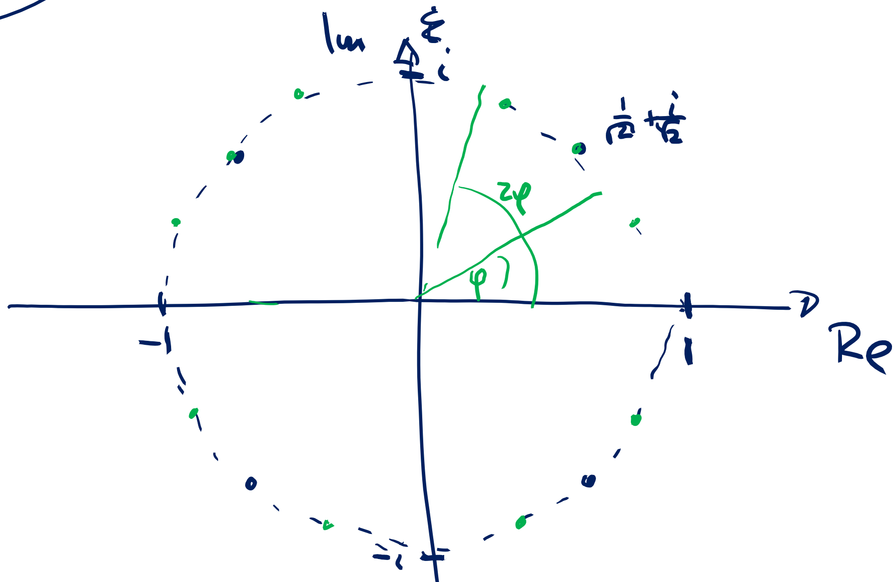
$1 \rightleftarrows 2$, x^2

$\{1\}$

$\{1, -1\}$

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

$\{1, -1, i, -i, \frac{1}{\sqrt{2}} + \frac{i}{\sqrt{2}}, -\frac{1}{\sqrt{2}} + \frac{i}{\sqrt{2}}, -\frac{1}{\sqrt{2}} - \frac{i}{\sqrt{2}}, \frac{1}{\sqrt{2}} - \frac{i}{\sqrt{2}}\}$



Choice of X

$x^N = 1$

- Select points $x_0, x_1, \dots, x_{N-1}$ to evaluate p and q in a clever way

Consider the N complex roots of unity:

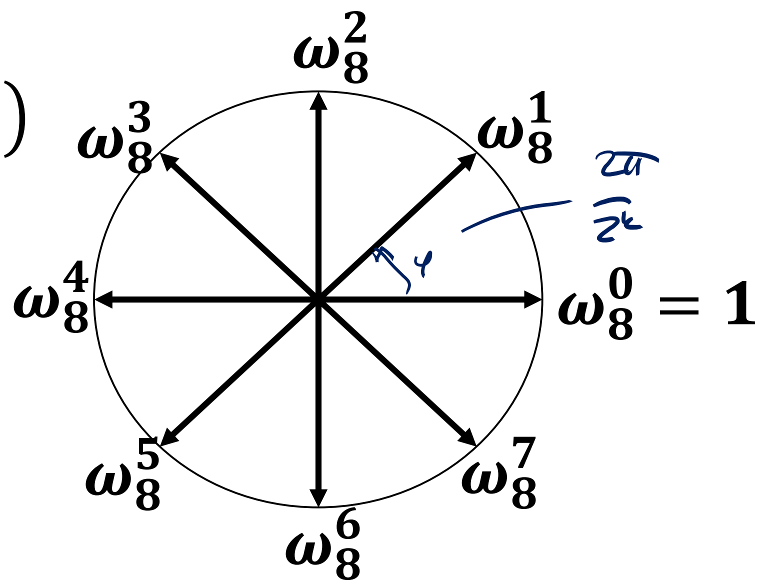
angle $= \frac{2\pi}{N}$

Principle root of unity: $\omega_N = e^{2\pi i / N}$

$$\left(i = \sqrt{-1}, \qquad e^{2\pi i} = 1\right)$$

Powers of ω_N (roots of unity):

$$1 = \omega_N^0, \omega_N^1, \dots, \omega_N^{N-1}$$



$y = \cos\varphi + i\sin\varphi = e^{i\varphi}$

Note: $\omega_N^k = e^{2\pi i k / N} = \cos\frac{2\pi k}{N} + i \cdot \sin\frac{2\pi k}{N}$

Properties of the Roots of Unity

- **Cancellation Lemma:**

  For all integers $n > 0$, $k \geq 0$, and $d > 0$, we have:

  φ

$$\omega_{dn}^{dk} = \omega_n^k, \qquad \omega_n^{k+n} = \omega_n^k$$

- **Proof:**

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