Exponentiation

- Given a and n, compute a^n.
- $a \times a \times \cdots \times a$ $\quad n$ times
- n-1 multiplications
- $a^{16} = (((a^2)^2)^2)^2$ only 4 multiplications
- $a^{11} = ((a^2)^2)^2 \times a^2 \times a$ only 5 multiplications
- Repeated squaring technique

Repeated Squaring

```
• Exp($a$, $n$) :
    • If $n$ is even
        • return ( Exp($a$, $n/2$) )$^2$
    • If $n$ is odd
        • return ( Exp($a$, $(n-1)/2$) )$^2 \times a$
    • If $n$ is 1
        • return $a$
```

Number of multiplications

$$T(n) \leq T(n/2) + 2$$

$$T(n) \leq 2 \log n$$

Another implementation

```
• Exp($a$, $n$) :
    • If $n$ is even
        • return ( Exp($a^2$, $n/2$) )
    • If $n$ is odd
        • return ( Exp($a^2$, $(n-1)/2$) ) $\times a$
    • If $n$ is 1
        • return $a$
```

Iterative implementation

- Input: a, n
- Initialize
  $a\_power\_n \leftarrow 1;$ // this will be a^n at the end
  $a\_two\_power \leftarrow a;$ // this will be $a^{2\^i}$ after i iterations
- while $(n > 0)$
    - If (n is odd) then $a\_power\_n \leftarrow a\_two\_power \times a\_power\_n;$
    - $a\_two\_power \leftarrow a\_two\_power \times a\_two\_power;$
    - $n \leftarrow n/2$ //integer part after division by 2 //or right-shift by 1 bit

Repeated squaring

- Does it give the minimum number of multiplications?
- What about a^{15} ?
  - can be done in 5 multiplications
- Given n, what the minimum number of multiplications required for a^n ? No easy answer.
- Can apply repeated squaring for other operations like Matrix powering.
- Apparently proposed by Pingala (200 BC ?).

Fibonacci numbers

- $F(n) = F(n-1) + F(n-2)$
- Used by Pingala to count the number of patterns of short and long vowels.
- Here again we are repeating the same operation n times.
- Can repeated squaring be used?
- Can we compute $F(n)$ in $O(\log n)$ arithmetic operations?

Fibonacci numbers

- $F(n) = F(n-1) + F(n-2)$
- if n is even
  - $F(n) = F(n/2)\, F(n/2-1) + F(n/2+1)F(n/2)$
    $= F(n/2)\, (2F(n/2+1) - F(n/2))$
  - $F(n+1) = F(n/2)\, F(n/2) + F(n/2+1)F(n/2+1)$
- similarly, if n is odd

Fibonacci numbers

$$\begin{pmatrix} F_3 \\ F_2 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} F_1 \\ F_0 \end{pmatrix}$$

$$\begin{pmatrix} F_5 \\ F_4 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} F_3 \\ F_2 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}^2 \begin{pmatrix} F_1 \\ F_0 \end{pmatrix}$$

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}^{n/2} \begin{pmatrix} F_1 \\ F_0 \end{pmatrix}$$

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n} \begin{pmatrix} F_1 \\ F_0 \end{pmatrix}$$

Integer Multiplication

- Addition: Adding two n bit numbers

  School method $O(n)$ time

- $O(n)$ time is necessary to write the output.

- Multiplication: multiplying two n bit numbers

  School method $O(n^2)$ time

- is $O(n^2)$ time is necessary?

Integer Multiplication

- Kolmogorov (1960) conjectured that $O(n^2)$ time is necessary.
- In a week, Karatsuba found $O(n^{1.59})$ time algorithm.
- Karatsuba quoted in Jeff Erickson's Algorithms
- "After the seminar I told Kolmogorov about the new algorithm and about the disproof of the n^2 conjecture. Kolmogorov was very agitated because this contradicted his very plausible conjecture. At the next meeting of the seminar, Kolmogorov himself told the participants about my method, and at that point the seminar was terminated."

Special case: Integer Squaring

- If we have a squaring algorithm, does it directly give us a multiplication algorithm?
- Input: n bit integer a
- Break it into two chunks $n\text{-}1$ bits and 1 bit
- $a = \varepsilon + 2\,b$
- $a^2 = \varepsilon^2 + 2\,b\,\varepsilon + b^2$
- $T(n) = T(n\text{-}1) + O(n) = O(n^2)$

Squaring: divide and conquer

- Input: n bit integer a
- Break it into two chunks: $n/2$ bits and $n/2$ bits
- $a = b + c\ 2^{n/2}$.
- $a^2 = b^2 + 2\ b\ c\ 2^{n/2} + c^2\ 2^n$
- Need to compute the multiplication bc recursively.
- How to compute bc with the square function?
- $2\ bc = (b+c)^2 - b^2 - c^2$

Squaring: divide and conquer

- Input: n bit integer a
- Break it into two chunks: $n/2$ bits and $n/2$ bits
- $a^2 = b^2 + (b^2 + c^2 - (b-c)^2) 2^{n/2} + c^2 2^n$
- Squaring an n bit number reduced to
  - squaring three $n/2$ bit numbers
  - and few additions and left shifts $O(n)$
- $T(n) = 3 T(n/2) + O(n)$

Running time analysis

- $T(n) = 3\ T(n/2) + O(n)$
    - $T(n) \le c\ n + 3\ T(n/2)$ for some constant c
    - $T(n) \le c\ n + 3cn/2 + 3^2\ T(n/2^2)$
    - $T(n) \le c\ n + 3cn/2 + 3^2\ cn/2^2 + 3^3\ T(n/2^3)$
    - $T(n) \le c\ n + 3cn/2 + 3^2\ cn/2^2 + \cdots + 3^{k-1}\ cn/2^{k-1} + 3^k\ T(n/2^k)$
    - Assuming $n = 2^k$ and $T(1) = 1$
    - $T(n) \le c\ n\ (1 + 3/2 + 3^2/2^2 + \cdots + 3^{k-1}/2^{k-1}) + 3^k$
    - $T(n) \le 2\ c\ n\ (3^k/2^k - 1) + 3^k \le (2c+1)\ 3^k = O(3^k) = O(3^{\log n}) = O(n^{\log 3})$

Karatsuba's multiplication

- Input: n bit integers a *and* b
- Break integers into two chunks: $n/2$ bits and $n/2$ bits
- $a = a_0 + a_1 2^{n/2}$.
- $b = b_0 + b_1 2^{n/2}$.
- $ab = a_0 b_0 + (a_0 b_1 + a_1 b_0) 2^{n/2} + a_1 b_1 2^n$
- Want to compute the three terms $a_0 b_0$, $(a_0 b_1 + a_1 b_0)$, $a_1 b_1$ with only three multiplications and a few additions
- $T(n) = 3 T(n/2) + O(n)$
- $T(n) = O(n^{\log 3}) = O(n^{1.585})$

Toom-Cook multiplication

- Break it into three chunks: $n/3$ bits each
- $a = a_0 + a_1\, 2^{n/3} + a_2\, 2^{2n/3}$
- $b = b_0 + b_1\, 2^{n/3} + b_2\, 2^{2n/3}$
- $ab = a_0 b_0 + (a_0 b_1 + a_1 b_0)\, 2^{n/3} + (a_0 b_2 + a_1 b_1 + a_2 b_0)\, 2^{2n/3}$

  $+ (a_1 b_2 + a_2 b_1)\, 2^{3n/3} + a_2 b_2\, 2^{4n/3}$
- In how many multiplications of n/3 bit numbers, can you find these five terms?

Toom-Cook multiplication

- $ab = a_0 b_0 + (a_0 b_1 + a_1 b_0)\, 2^{n/3} + (a_0 b_2 + a_1 b_1 + a_2 b_0)\, 2^{2n/3}$

  $+ (a_1 b_2 + a_2 b_1)\, 2^{3n/3} + a_2 b_2\, 2^{4n/3}$
- In how many multiplications of n/3 bit numbers, can you find these five terms?
- If six multiplications
  - $T(n) = 6\, T(n/3) + O(n) \Longrightarrow T(n) = O(n^{(\log 6)/(\log 3)}) = O(n^{1.63})$
- If five multiplications
  - $T(n) = 5\, T(n/3) + O(n) \Longrightarrow T(n) = O(n^{(\log 5)/(\log 3)}) = O(n^{1.46})$

Toom-Cook multiplication

- Homework: find these five terms using five multiplications and a few additions
    - $a_0 b_0$
    - $a_0 b_1 + a_1 b_0$
    - $a_0 b_2 + a_1 b_1 + a_2 b_0$
    - $a_1 b_2 + a_2 b_1$
    - $a_2 b_2$

Toom-Cook multiplication

- Homework: find these five terms using five square operations and a few additions
  - P^2
  - PQ
  - $2PR + Q^2$
  - QR
  - R^2

Integer multiplication history

- Karatsuba: break integers into two parts: $O(n^{1.585})$
- Toom-Cook: break integers into three parts: $O(n^{1.46})$
- What if break into more parts?
  - can get better and better time complexity by increasing the number of parts
  - for k parts, we will get $O(k^2 n^{\log (2k-1) / \log k})$
  - we can get $O(n^{1+\varepsilon})$ for any constant $\varepsilon > 0$

Integer multiplication history

- for k parts, we will get $O(k^2 n^{\log(2k-1) / \log k})$
- we can get $O(n^{1+\varepsilon})$ for any constant $\varepsilon > 0$
- Exercise: how many parts to break into for $O(n^{1.1})$
- Theoretically faster, but not necessarily faster in practice due to large constants.
- For multiplying 64 bit integers, Karatsuba may not be faster than school method. A combination of the two may be faster.

Integer multiplication history

- [1960] Karatsuba $O(n^{1.585})$
- [1963, 1966] Toom-Cook: $O(n^{1.46})$
- [1981] Donald Knuth: $O(n\, 2^{\sqrt{(2 \log n)}} \log n)$
- [1971] Schönhage–Strassen: $O(n \log n \log\log n)$
- [2007, 2008] Fürer, De-Kurur-Saha-Saptharishi: $O(n \log n\, 2^{\log^* n})$
- [2019] Harvey-van der Hoeven: $O(n \log n)$
- Conjecture: $O(n \log n)$ can not be improved

Matrix multiplication

- Input: $n \times n$ matrices *A and B*
- Break matrices into four parts: $n/2 \times n/2$ each

$$A = \begin{pmatrix} A_0 & A_1 \\ A_2 & A_3 \end{pmatrix} \qquad B = \begin{pmatrix} B_0 & B_1 \\ B_2 & B_3 \end{pmatrix}$$

$$AB = \begin{pmatrix} A_0B_0 + A_1B_2 & A_0B_1 + A_1B_3 \\ A_2B_0 + A_3B_2 & A_2B_1 + A_3B_3 \end{pmatrix}$$

- Want to compute the four matrices with only seven multiplications and a few additions

Polynomial multiplication

- Input: degree d polynomials P *and* Q
- $P(x) = p_0 + p_1 x + p_2 x^2 + \cdots + p_d x^d.$
- $Q(x) = q_0 + q_1 x + q_2 x^2 + \cdots + q_d x^d.$
- $P(x)\, Q(x) = p_0 + (p_0q_1+p_1q_0)x + \cdots + p_dq_d\, x^{2d}.$
- Assume integer multiplication in unit time.
- School multiplication $O(d^2)$ time.
- Can we do better?

Next week

- Quiz: Wednesday, Jan 31, 8:30 AM, open notes
- Polynomial multiplication, Fast Fourier transform
- Puzzle: secret sharing
- Share secret among n parties
  If any k of them come together, the secret can be reconstructed.

Polynomial multiplication

- $A(x) = a_0 + a_1\, x + a_2\, x^2 + \cdots + a_{d\text{-}1}\, x^{d\text{-}1}$
- $B(x) = b_0 + b_1\, x + b_2\, x^2 + \cdots + b_{d\text{-}1}\, x^{d\text{-}1}$
- $A(x)B(x)$
  $= a_0\, b_0 + (a_0\, b_1 + a_1\, b_0)\, x + (a_0\, b_2 + a_1\, b_1 + a_2\, b_0)\, x^2$
  $+ \cdots + a_{d\text{-}1}\, b_{d\text{-}1}\, x^{2d\text{-}2}$
- $$A(x)B(x) = \sum_{j=0}^{2d-2} x^j \left(\sum_{i=0}^{j} a_i b_{j-i} \right)$$

Polynomial multiplication

- $A(x)B(x)$
  $= a_0 b_0 + (a_0 b_1 + a_1 b_0)\, x + (a_0 b_2 + a_1 b_1 + a_2 b_0)\, x^2$
  $+ \cdots + a_{d-1} b_{d-1}\, x^{2d-2}$
- Assuming unit cost arithmetic operations, what is the time complexity?
- Naive algorithm $O(d^2)$
- Can we use Karatsuba's idea for polynomial multiplication?

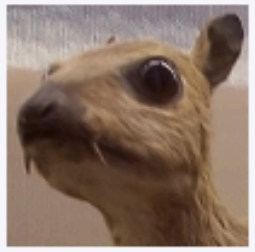
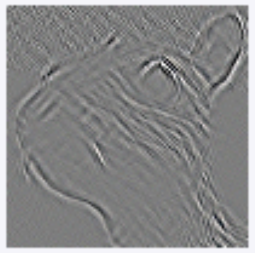
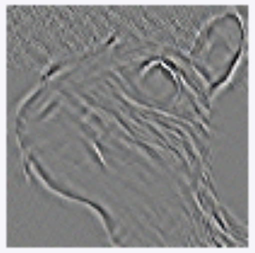
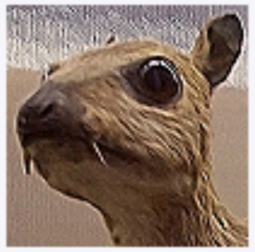
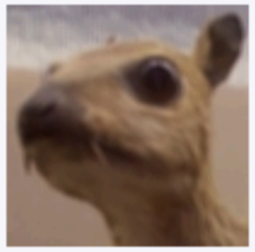
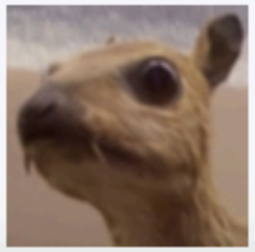
Convolution

- $a = (a_0, a_1, a_2, \ldots, a_{m\text{-}1}) \in R^m$
- $b = (b_0, b_1, b_2, \ldots, b_{n\text{-}1}) \in R^n$
- $a * b = (a_0 b_0, (a_0 b_1 + a_1 b_0), (a_0 b_2 + a_1 b_1 + a_2 b_0),$

  $\ldots, (\sum_i a_i b_{j-i}) \ \ldots, \ a_{m\text{-}1} b_{n\text{-}1}) \in R^{m+n\text{-}1}$
- Dot product with a sliding window

Applications of Convolution

- Signal processing
  - Smoothening of noisy data
  - Covid cases per day: take seven day averages
- Image processing: 2D convolution (bivariate polynomial multiplication)
- Convolutional neural networks

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Ridge or edge detection	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

- Source: https://en.wikipedia.org/wiki/Kernel_(image_processing)

Applications of Convolution

- Probability distribution of a sum of two random variables
- Dice with

Outcome	1	2	3	4	5	6
Probability	0.2	0.3	0.1	0.1	0.2	0.1

- Sum of two such dice

Outcome	1	2	3	4	5	6	7	. . .
Probability	0	0.04	0.12 ….					

Polynomial multiplication / Convolution

- Can we get faster than the naive $O(d^2)$ algorithm?
- Different representations of a polynomial
    - coefficients
    - roots
    - evaluations
- Claim: given d evaluations, there is a unique degree $d-1$ polynomial satisfying those evaluations.

Unique polynomial with d evaluations

- Claim: given d evaluations, there is a unique degree $d\text{-}1$ polynomial satisfying those evaluations.
- Proof:
  - Suppose there are two different degree $d\text{-}1$ polynomials $P(x)$ and $Q(x)$ which have the same d evaluations.
  - Define $R(x) = P(x) - Q(x)$.
  - Then $R(x)$ is zero at these d points
  - But, a degree $d\text{-}1$ polynomial cannot have d roots. Contradiction.

Roots of a polynomial

- Claim: A (nonzero) degree d-1 polynomial can have at most d-1 roots
- Proof: We prove it by induction.
- Base case: degree 1 polynomial has exactly one root.
- Induction hypothesis: a degree d-2 polynomial has at most d-2 roots
- Induction step: Let $P(x)$ be a polynomial of degree d-1.
  - Let α be a root of $P(x)$. Then x-α divides $P(x)$.
  - Let $P(x) = (x\text{-}\alpha)\ Q(x)$. If $\mathrm{P}(\beta) = 0$ for some $\beta \neq \alpha$, then $\mathrm{Q}(\beta) = 0$.
  - Hence, all other roots of $P(x)$ are also roots of $Q(x)$.
  - By induction hypothesis, $Q(x)$ has at most d-2 roots.
  - Hence, $P(x)$ has at most $d\text{-}2+1 = d\text{-}1$ roots.

Polynomial representations

- Computations with different representations

	Addition	Multiplication
Coefficients	$O(d^2)$	$O(d^2)$
Roots	$?$	$O(d)$
Evaluations	$O(d)$	$O(d)$

- But, can we convert between different representations?

Polynomial Representations

- Given d coefficients, how much time needed to compute evaluations of the polynomial at d points?
- Naively, each evaluation will need $O(d)$ multiplications and additions
- Overall time $O(d^2)$
- Can evaluation at one point help with evaluating at another point?

Coefficients to evaluations

- $A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{d-1} x^{d-1}$
- $A(1) = a_0 + a_1 + a_2 + \cdots + a_{d-1}$
- $A(-1) = a_0 - a_1 + a_2 - \cdots - a_{d-1}$
- Total additions = $2d-2$
- First compute
  even-sum = $a_0 + a_2 + a_4 + \cdots + a_{d-2}$
  odd-sum = $a_1 + a_3 + a_5 + \cdots + a_{d-1}$
- $A(1) = \textit{even-sum} + \textit{odd-sum}$
- $A(-1) = \textit{even-sum} - \textit{odd-sum}$
- Total additions = d

Coefficients to evaluations

- Similar trick with $A(\alpha)$ and $A(-\alpha)$
- $A(x) = a_0 + a_1\ x + a_2\ x^2 + \cdots + a_{d-1}\ x^{d-1}$
- $A_{even}(x) = a_0 + a_2\ x + a_4\ x^2 + \cdots + a_{d-2}\ x^{d/2-1}$
- $A_{odd}(x) = a_1 + a_3\ x + a_5\ x^2 + \cdots + a_{d-1}\ x^{d/2-1}$
- $A(\alpha) = A_{even}(\alpha^2) + \alpha \cdot A_{odd}(\alpha^2)$
- $A(-\alpha) = A_{even}(\alpha^2) - \alpha \cdot A_{odd}(\alpha^2)$
- Two evaluations of degree $d\text{-}1$ polynomial
  $\rightarrow$ Two evaluations degree $d/2\text{-}1$ polynomials

Coefficients to evaluations

- $A(\alpha) = A_{even}(\alpha^2) + \alpha \cdot A_{odd}(\alpha^2)$
- $A(-\alpha) = A_{even}(\alpha^2) - \alpha \cdot A_{odd}(\alpha^2)$
- Evaluations of $A(x)$ at
  $\alpha, -\alpha, \beta, -\beta, \gamma, -\gamma, \ldots.$
- $\rightarrow$ Evaluations of $A_{even}(x)$ and $A_{odd}(x)$ at
  $\alpha^2, \beta^2, \gamma^2, \ldots.$
- Total work reduced by half
- Can we further apply this trick?
- We will need $\alpha^2 = - \beta^2$?

Roots of unity

- Evaluations of $A(x)$ at
  $1, -1, i, -i$
- $\rightarrow$ Evaluations of $A_{even}(x)$ and $A_{odd}(x)$ at
  $1,\ i^2 = -1$
- $\rightarrow$ Evaluations of $A_{even\text{-}even}(x)$, $A_{even\text{-}odd}(x)$, $A_{odd\text{-}even}(x)$, $A_{odd\text{-}odd}(x)$ at
  1
- To generalize assume $d = 2^t$
- We will start with *d-th* roots of unity.

Roots of unity

- $a + i\,b = r\,e^{i\theta}$
- $r = \sqrt{(a^2 + b^2)}$ is the absolute value
- $\theta = tan^{-1}(b/a)$ is the angle with real axis
- $e^{i\pi} = -1$
- Primitive d-th root of unity $= \omega = e^{2\pi i/d}$
    - All d-th roots of unity $= 1, \omega, \omega^2, \omega^3, \ldots., \omega^{d-1}$

Properties of roots of unity

- Primitive d-th root of unity = $\omega = e^{2\pi i / d}$
    - $\omega^{d/2} = e^{2\pi i / 2} = -1$
    - All d-th roots of unity = $1, \omega, \omega^2, \omega^3, \ldots, \omega^{d-1}$
    - All $(d/2)$-th roots of unity = $1, \omega^2, \omega^4, \omega^6, \ldots, \omega^{2d-2}$
    - $1 + \omega + \omega^2 + \omega^3 + \cdots + \omega^{d-1} = 0$
    - $1 + \omega^j + \omega^{2j} + \omega^{3j} + \cdots + \omega^{(d-1)j} = 0$ for $0 < j < d$

Discrete Fourier Transform

- Evaluate a degree d-1 polynomial at d-th roots of unity
- Evaluations of $A(x)$ at
  $1, \omega, \omega^2, \omega^3, \ldots, \omega^{d-1}$ (d-th roots)
- *Given* $a_0, a_1, a_2, \cdots, a_{d-1}$
- Output
  $a_0 + a_1\, 1 + a_2\, 1^2 + \cdots + a_{d-1}\, 1^{d-1}$

  $a_0 + a_1\, \omega + a_2\, \omega^2 + \cdots + a_{d-1}\, \omega^{d-1}$

  $a_0 + a_1\, \omega^2 + a_2\, \omega^4 + \cdots + a_{d-1}\, \omega^{2d-2}$

  $a_0 + a_1\, \omega^{d-1} + a_2\, \omega^{2d-2} + \cdots + a_{d-1}\, \omega^{2d-1}$

Fourier Transform

- $F(k) = \int_{-\infty}^{\infty} f(x)\ sin(2\pi kx)\ dx$
- $G(k) = \int_{-\infty}^{\infty} f(x)\ cos(2\pi kx)\ dx$
- $G(k) + iF(k) = \int_{-\infty}^{\infty} f(x)\ e^{2\pi kx}\ dx$

Fast Fourier Transform

- Evaluations of $A(x)$ at $1, \omega, \omega^2, \omega^3, \ldots, \omega^{d-1}$
- Assume d is a power of 2
- $A_{even}(x) = a_0 + a_2\, x + a_4\, x^2 + \cdots + a_{d-2}\, x^{d/2-1}$
- $A_{odd}(x) = a_1 + a_3\, x + a_5\, x^2 + \cdots + a_{d-1}\, x^{d/2-1}$
- $-1 = \omega^{d/2}$
- $-\omega = \omega^{1+d/2}$

  ....

- $A(\alpha) = A_{even}(\alpha^2) + \alpha \cdot A_{odd}(\alpha^2)$
- $A(-\alpha) = A_{even}(\alpha^2) - \alpha \cdot A_{odd}(\alpha^2)$

- $-\omega^{d/2-1} = \omega^{d/2+d/2-1} = \omega^{d-1}$
- Squares of the d-th roots : $1, \omega^2, \omega^4, \ldots, \omega^{2(d/2-1)}$ ($d/2$-th roots)

Fast Fourier Transform
Cooley-Tukey 1965

- Evaluations of $A(x)$
  at $1, \omega, \omega^2, \omega^3, \ldots, \omega^{d-1}$ (d-th roots)
- Recursively evaluate $A_{even}(x)$ and $A_{odd}(x)$
  at $1, \omega^2, \omega^4, \omega^6, \ldots, \omega^{d-2}$ ($d/2$-th roots)
- For $0 \leq k \leq d/2-1$
  - $A(\omega^k) = A_{even}(\omega^{2k}) + \omega^k \cdot A_{odd}(\omega^{2k})$
  - $A(\omega^{k+d/2}) = A_{even}(\omega^{2k}) - \omega^k \cdot A_{odd}(\omega^{2k})$

Fast Fourier Transform

- Let $T(d)$ be the time to evaluate a degree d-1 polynomial at d-th roots of unity.
- For $0 \leq k \leq d/2\text{-}1$
  - $A(\omega^k) = A_{even}(\omega^{2k}) + \omega^k \cdot A_{odd}(\omega^{2k})$
  - $A(\omega^{k+d/2}) = A_{even}(\omega^{2k}) - \omega^k \cdot A_{odd}(\omega^{2k})$
- $T(d) = 2\ T(d/2) + O(d)$
- $T(d) = O(d \log d)$

Polynomial multiplicaiton / convolution

- Coefficients $\rightarrow$ Evaluations (FFT)
  - O(*d log d*)
- Pointwise multiply evaluations of the two polynomials
  - O(*d*)
- Evaluations $\rightarrow$ Coefficients (inverse FFT)
  - Inverse FFT is just FFT with ω replaced by ω^{-1}
  - O(*d log d*)

Inverse FFT

- Inverse FFT is just FFT with ω replaced by $\omega^{-1} = \omega^{d-1}$

$$\boxed{\begin{aligned}
e_0 &= a_0 + a_1\, 1 + a_2\, 1^2 + \cdots + a_{d-1}\, 1^{d-1} \\
e_1 &= a_0 + a_1\, \omega + a_2\, \omega^2 + \cdots + a_{d-1}\, \omega^{d-1} \\
&\ldots\ldots \\
e_{d-1} &= a_0 + a_1\, \omega^{d-1} + a_2\, \omega^{2d-2} + \cdots + a_{d-1}\, \omega^{2d-1}
\end{aligned}}$$

$$\boxed{\begin{aligned}
\mathrm{d}a_0 &= e_0 + e_1\, 1 + e_2\, 1^2 + \cdots + e_{d-1}\, 1^{d-1} \\
\mathrm{d}a_1 &= e_0 + e_1\, \omega^{-1} + e_2\, \omega^{-2} + \cdots + e_{d-1}\, \omega^{-d+1} \\
&\ldots\ldots \\
\mathrm{d}a_{d-1} &= e_0 + e_1\, \omega^{-d+1} + e_2\, \omega^{-2d+2} + \cdots + e_{d-1}\, \omega^{-2d+1}
\end{aligned}}$$

FFT Implementation issues

- Can we really do additions/multiplications with roots of unity in constant time?
  - approximations: how many bits sufficient?
  - Requires careful implementation.
  - modular arithmetic: work modulo a large enough prime, such that d-th roots of unity exist.
- Is it fast in practice?
  - Iterative implementation, which is memory efficient
  - FFT convolution faster than direct convolution for $d=128$

Fast integer multiplication

- Split the integers into d parts each having t bits
- $a = a_0 + a_1\, 2^t + a_2\, 2^{2t} + \cdots + a_{d\text{-}1}\, 2^{(d\text{-}1)t}$
- $b = b_0 + b_1\, 2^t + b_2\, 2^{2t} + \cdots + b_{d\text{-}1}\, 2^{(d\text{-}1)t}$
- Define
  - $A(x) = a_0 + a_1\, x + a_2\, x^2 + \cdots + a_{d\text{-}1}\, x^{d\text{-}1}$
  - $B(x) = b_0 + b_1\, x + b_2\, x^2 + \cdots + b_{d\text{-}1}\, x^{d\text{-}1}$
- Multiply as polynomials (using FFT) and put $x = 2^t$