Agenda

We've done

- Asymptotic Analysis
- Solving Recurrence Relations

Now

- Designing Algorithms with the Divide and Conquer Method

The Basic Idea

- Divide: Partition the problem into smaller ones
- Conquer: Recursively solve the smaller problems
- Combine: Use solutions to smaller problems to give solution to larger problem

Puzzle

Given an array $A[1,\dots,n]$ of real numbers. Report the largest sum of numbers in a (contiguous) sub-array of A.

Merge Sort – The Canonical Example of D&C

Given an array $A[1,\ldots,n]$ of numbers, sort it in ascending order

- Divide: $A[1,\ldots,n/2]$, $A[n/2+1,\ldots,n]$
- Conquer: Sort $A[1,\ldots,n/2]$, sort $A[n/2+1,\ldots,n]$
- Combine: from two sorted sub-array, *somehow* "merge" them into a sorted array (see posted demo)
- Running time:

$$T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = O(n \lg n)$$

- The key is the $\Theta(n)$-merge step.

Counting Inversions: Problem Definition

- Input: an array $A[1..n]$ of distinct integers
- Output: the number of pairs (i,j) such that $i < j$, $A[i] > A[j]$
- Applications: numerous
  - Voting theory
  - Collaborative filtering
  - Sensitivity analysis of Google's ranking function
  - Rank aggregation for meta-searching on the Web
  - Non-parametric statistics (Kendalls' Tau function)
- Brute force: $O(n^2)$
- Can we do better?

Divide and Conquer

- Divide: $A_1 = A[1,\dots,n/2]$, $A_2 = A[n/2+1,\dots,n]$
- Conquer: $a_i =$ number of inversions in A_i, $i = 1,2$
- Combine: $a =$ number of "inter-inversions," i.e.

$$a = \#\{(i,j) \mid i \le n/2, j > n/2, A[i] > A[j]\}$$

  Return $a_1 + a_2 + a$.
- Main question: how to combine efficiently?
  - Obvious approach: the combine step takes $\Theta(n^2)$

$$T(n) = 2T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^2)$$

  - Non-obvious: the combine step takes $\Theta(n)$ (see demo)

$$T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \lg n)$$

Multiplying Large Integers: Problem Definition

- Let i and j be two n-bit integers, compute ij.
- Straightforward multiplication takes $\Theta(n^2)$
- Naive D&C:

$$\begin{aligned} i &= a2^{n/2} + b \\ j &= x2^{n/2} + y \end{aligned}$$

$$ij = ax2^n + (ay + bx)2^{n/2} + by$$

  Running time:

$$T(n) = 4T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n^2).$$

Observation

Addition and shift take $\Theta(n)$, hence we want to reduce the number of (recursive) multiplications

(Smart) Divide and Conquer

- Want: compute three terms $ax, by, ay + bx$ using less than 4 multiplications.
- Observation:

$$\begin{aligned} P_1 &= ax \\ P_2 &= by \\ P_3 &= (a+b)(x+y) = (ay+bx) + ax + by \\ ay + bx &= P_3 - P_1 - P_2 \end{aligned}$$

- Immediately we have a D&C algorithm with running time

$$T(n) = 3T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.59})$$

Matrix Multiplication: Problem Definition

- $\mathbf{X}$ and $\mathbf{Y}$ are two $n \times n$ matrices. Compute $\mathbf{XY}$.
- Straightforward method takes $\Theta(n^3)$.
- Naive D&C:

$$\begin{aligned} \mathbf{XY} &= \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{S} & \mathbf{T} \\ \mathbf{U} & \mathbf{V} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{AS} + \mathbf{BU} & \mathbf{AT} + \mathbf{BV} \\ \mathbf{CS} + \mathbf{DU} & \mathbf{CT} + \mathbf{DV} \end{bmatrix} \end{aligned}$$

$$T(n) = 8T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^3)$$

Smart D&C: Strassen's Algorithm

- Idea: reduce the number of multiplications to be < 8. E.g.,

$$T(n) = 7T(n/2) + \Theta(n^2) \Rightarrow T(n) = n^{\log_2 7} = o(n^3)$$

- Want: 4 terms (in lower-case letters for easy reading)

$$as + bu$$

$$at + bv$$

$$cs + du$$

$$ct + dv$$

Strassen's Brilliant Insight

$$\begin{array}{lclcl}
p_1 & = & (a-c)(s+t) & = & \mathbf{as} + \mathbf{at} - \mathbf{cs} - \mathbf{ct} \\
p_2 & = & (b-d)(u+v) & = & \mathbf{bu} + \mathbf{bv} - \mathbf{du} - \mathbf{dv} \\
p_3 & = & (a+d)(s+v) & = & \mathbf{as} + \mathbf{dv} + av + ds \\
p_4 & = & a(t-v) & = & \mathbf{at} - av \\
p_5 & = & (a+b)v & = & \mathbf{bv} + av \\
p_6 & = & (c+d)s & = & \mathbf{cs} + ds \\
p_7 & = & d(u-s) & = & \mathbf{du} - ds
\end{array}$$

The rest is simply ... magical

$$\begin{array}{rcl}
as + bu & = & p_2 + p_3 - p_5 + p_7 \\
at + bv & = & p_4 + p_5 \\
cs + du & = & p_6 + p_7 \\
ct + dv & = & p_3 + p_4 - p_1 - p_6
\end{array}$$

Quick Sort: Basic Idea

- Input: array A, two indices p, q
- Output: same array with $A[p, \ldots, q]$ sorted
- Idea: use divide & conquer
  - Divide: rearrange $A[p, \ldots, q]$ so that for some r in between p and q,

$$\begin{aligned} A[i] &\leq A[r] \;\; \forall i = p, \ldots, r-1 \\ A[r] &\leq A[j] \;\; \forall j = r+1, \ldots, q \end{aligned}$$

    Compute r as part of this step.
  - Conquer: Quicksort($A[p, \ldots, r-1]$), and Quicksort($A[r+1, \ldots, q]$)
  - Combine: Nothing

Quicksort: Pseudo code

Quicksort(A, p, q)

```
if p < q then
    r ← Partition(A, p, q)
    Quicksort(A, p, r − 1)
    Quicksort(A, r + 1, q)
end if
```

Rearrange $A[p..q]$: partitioning

	i	p,j							q		
...		3	1	8	5	6	2	7	4		...
		p,i	j						q		
...		3	1	8	5	6	2	7	4		...
		p	i	j					q		
...		3	1	8	5	6	2	7	4		...
		p	i		j				q		
...		3	1	8	5	6	2	7	4		...
		p	i			j			q		
...		3	1	8	5	6	2	7	4		...
		p	i				j		q		
...		3	1	8	5	6	2	7	4		...
...		3	1	2	5	6	8	7	4		...
...		3	1	2	5	6	8	7	4		...
...		3	1	2	4	6	8	7	5		...

Partitioning: pseudo code

The following code partitions $A[p..q]$ around $A[q]$

Partition(A, p, q)

$x \leftarrow A[q]$
$i \leftarrow p - 1$
for $j \leftarrow p$ **to** q **do**
    if $A[j] \leq x$ **then**
        swap $A[i+1]$ and $A[j]$
        $i \leftarrow i + 1$
    end if
end for
return i

Question

How would you partition around $A[m]$ for some m: $p \leq m \leq q$?

Worst case running time

- Let $T(n)$ be the worst-case running time of Quicksort.
- Then $T(n) = \Omega(n^2)$ (why?)
- We shall show $T(n) = O(n^2)$, implying $T(n) = \Theta(n^2)$.

$$T(n) = \max_{0 \le r \le n-1} (T(r) + T(n-r-1)) + \Theta(n)$$

$T(n) = O(n^2)$ follows by induction.

Quicksort is "most often" quicker than the worst case

Worst-case partitioning:

$$T(n) = T(n-1) + T(0) + \Theta(n) = T(n-1) + \Theta(n)$$

yielding $T(n) = O(n^2)$.
Best-case partitioning:

$$T(n) \approx 2T(n/2) + \Theta(n)$$

yielding $T(n) = O(n \lg n)$.
Somewhat balanced partitioning:

$$T(n) \approx T\left(\frac{n}{10}\right) + T\left(9\frac{n}{10}\right) + \Theta(n)$$

yielding $T(n) = O(n \lg n)$ (recursion-tree).

Average-case running time: a sketch

Claim

The running time of Quicksort is proportional to the number of comparisons

Let M_n be the expected number of comparisons (what's the sample space?).
Let X be the random variable counting the number of comparisons.

$$
\begin{aligned}
M_n = E[X] &= \sum_{j=1}^{n} E[X \mid A[q] \text{ is the } j\text{th least number}]\frac{1}{n} \\
&= \frac{1}{n}\sum_{j=1}^{n}\left(n-1+M_{j-1}+M_{n-j}\right) \\
&= n-1+\frac{2}{n}\sum_{j=0}^{n-1} M_j
\end{aligned}
$$

Randomized Quicksort

Randomized-Quicksort(A,p,q)

if $p < q$ **then**
    $r \leftarrow$ Randomized-Partition(A,p,q)
    Randomized-Quicksort$(A,p,r-1)$
    Randomized-Quicksort$(A,r+1,q)$
end if

Randomized-Partition(A,p,q)

pick m at random between p,q
swap $A[m]$ and $A[q]$
$x \leftarrow A[q]$; $i \leftarrow p-1$
for $j \leftarrow p$ **to** q **do**
    if $A[j] \le x$ **then**
        swap $A[i+1]$ and $A[j]$; $i \leftarrow i+1$
    end if
end for
return i

The Selection Problem: Definition

- The ith order statistic of a set of n numbers is the ith smallest number
- The median is the $\lfloor n/2 \rfloor$th order statistic
- Selection problem: find the ith order statistic as fast as possible

Conceivable that the running time is proportional to the number of comparisons.

- Find a way to determine the 2nd order statistic using as few comparisons as possible
- How about the 3rd order statistic?

Selection in Worst-case Linear Time

- Input: $A[p, \dots, q]$ and i, $1 \le i \le q - p + 1$
- Output: the ith order statistic of $A[p, \dots, q]$
- Idea: same as Randomized-Selection, but also try to guarantee a good split.
  - Find $A[m]$ which is not too far left nor too far right
  - Then, split around $A[m]$

The idea is from: Manuel Blum, Vaughan Pratt, Robert E. Tarjan, Robert W. Floyd, and Ronald L. Rivest, **"Time bounds for selection."** *Fourth Annual ACM Symposium on the Theory of Computing (Denver, Colo., 1972).* Also, J. Comput. System Sci. 7 (1973), 448–461.

Linear-Selection: Pseudo-Code

Linear-Select(A, i)

1: "Divide" n elements into $\lceil \frac{n}{5} \rceil$ groups,
- $\lfloor \frac{n}{5} \rfloor$ groups of size 5, and
- $\lceil \frac{n}{5} \rceil - \lfloor \frac{n}{5} \rfloor$ group of size $n - 5\lfloor \frac{n}{5} \rfloor$

2: Find the median of each group
3: Find x: the median of the medians by calling Linear-Select recursively
4: Swap $A[m]$ with $A[n]$, where $A[m] = x$
5: $r \leftarrow$ Partition$(A, 1, n)$
6: **if** $r = i$ **then**
7:     return $A[r]$
8: **else**
9:     recursively go left or right accordingly
10: **end if**

Linear-Select: Analysis

- $T(n)$ denotes running time
- Lines 1 & 2: $\Theta(n)$
- Line 3: $T(\lceil \frac{n}{5} \rceil)$
- Lines 4, 5: $\Theta(n)$
- Lines 6-10: at most $T(f(n))$, where $f(n)$ is the larger of two numbers:
  - number of elements to the left of $A[r]$,
  - number of elements to the right of $A[r]$

$f(n)$ could be shown to be at most $\frac{7n}{10} + 6$, hence

$$T(n) \leq \begin{cases} \Theta(1) & \text{if } n \leq 71 \\ T(\lceil \frac{n}{5} \rceil) + T(\lfloor \frac{7n}{10} + 6 \rfloor) + \Theta(n) & \text{if } n > 71 \end{cases}$$

Induction gives $T(n) = O(n)$

Fast Fourier Transform: Motivations

- Roughly, Fourier Transforms allow us to look at a function in two different ways
- In (analog and digital) communication theory:
  - time domain $\xrightarrow{FT}$ frequency domain
  - time domain $\xleftarrow{FT^{-1}}$ frequency domain
  - For instance: every (well-behaved) T-periodic signal can be written as a sum of sine and cosine waves (sinusoids).

Fourier Series of Periodic Functions

$$x(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos(2\pi n f_0 t) + \sum_{n=1}^{\infty} b_n \sin(2\pi n f_0 t)$$

$f_0 = 1/T$ is the fundamental frequency.
Euler's formulas:

$$\begin{aligned}
\frac{a_0}{2} &= f_0 \int_{t_0}^{t_0+T} x(t)dt \\
\frac{a_n}{2} &= f_0 \int_{t_0}^{t_0+T} x(t)\cos(2\pi n f_0 t)dt \\
\frac{b_n}{2} &= f_0 \int_{t_0}^{t_0+T} x(t)\sin(2\pi n f_0 t)dx
\end{aligned}$$

Problem

Find a natural science without an Euler's formula

Continuous Fourier Transforms of Aperiodic Signals

- Basically, just a limit case of Fourier series when $T \to \infty$
- Applications are numerous: DSP, DIP, astronomical data analysis, seismic, optics, acoustics, etc.
- Forward Fourier transform

$$F(\nu) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i \nu t} dt.$$

- Inverse Fourier transform

$$f(t) = \int_{-\infty}^{\infty} F(\nu) e^{2\pi i t \nu} d\nu.$$

(Physicists like to use the *angular frequency* $\omega = 2\pi\nu$)

Discrete Fourier Transforms

- Computers can't handle continuous signals $\Rightarrow$ discretize it
- Sampling at n places:

$$f_k = f(t_k), \quad t_k = k\Delta, \quad k = 0, \dots, n-1$$

- DFT (continuous $\to$ discrete, integral $\to$ sum)

$$F_m = \sum_{k=0}^{n-1} f_k (e^{-2\pi i m/n})^k, \; 0 \leq m \leq n-1$$

- DFT^{-1}:

$$f_k = \frac{1}{n} \sum_{m=0}^{n-1} F_m (e^{2\pi i k/n})^m, \; 0 \leq k \leq n-1$$

Fundamental Problem

DFT and DFT^{-1} efficiently

Another Motivation: Operations on Polynomials

- A polynomial $A(x)$ over $\mathbb{C}$:

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1}x^{n-1} = \sum_{j=0}^{n-1} a_j x^j.$$

- $A(x)$ is of degree k if a_k is the highest non-zero coefficient. E.g,

$$B(x) = 3 - (2-4i)x + x^2 \text{ has degree } 2.$$

- If $m >$ degree(A), then m is called a degree bound of the polynomial. E.g. , $B(x)$ above has degree bounds $3, 4, \dots$

Common Operations on Polynomials

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

Addition

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

Multiplication

$$C(x) = A(x)B(x) = c_0 + c_1 x + \cdots + c_{2n-2}x^{2n-2}$$

$$c_j = \sum_{j=0}^{k} a_j b_{k-j}, \quad 0 \le k \le 2n-2$$

These are important problems in scientific computing.

Polynomial Representations

$$A(x) = a_0 + a_1 x + \dots a_{n-1} x^{n-1}.$$

Coefficient representation: a vector $\mathbf{a}$

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

Point-value representation: a set of point-value pairs

$$\{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$$

where the x_j are distinct, and $y_j = A(x_j), \forall j$

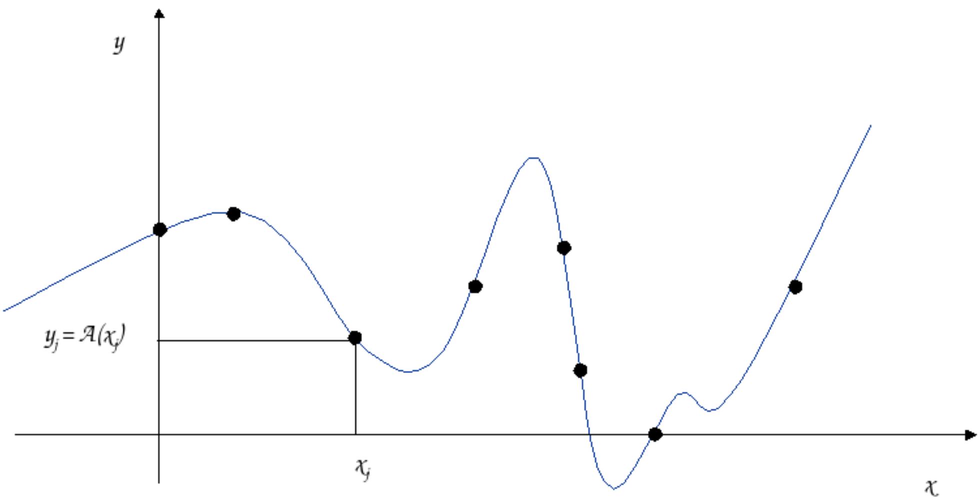
Question

How do we know that a set of point-value pairs represent a unique polynomial? What if there are two polynomials with the same set of point-value pairs?

Uniqueness of P-V Representation, First Proof

Fundamental Theorem of Algebra (Gauss' Ph.D thesis) A degree-n polynomial over $\mathbb{C}$ has n complex roots

Corollary A degree-$(n-1)$ polynomial is uniquely specified by n different values of x



Uniqueness of P-V Representation, Second Proof

Proof.

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

The matrix is called the Vandermonde matrix $V(x_0, \dots, x_{n-1})$, which has non-zero determinant

$$\det(V(x_0, \dots, x_{n-1})) = \prod_{p<q}(x_p - x_q).$$

□

Pros and Cons of Coefficient Representation

- Computing the sum $A(x) + B(x)$ takes $\Theta(n)$,
- Evaluating $A(x_k)$ take $\Theta(n)$ with Horner's rule

$$A(x_k) = a_0 + x_k(a_1 + x_k(a_2 + \cdots + x_k(a_{n-2} + x_k a_{n-1}) \dots)$$

  (we assume $+$ and $*$ of numbers take constant time)
- Very convenient for user interaction
- Computing the product $A(x)B(x)$ takes $\Theta(n^2)$, however

Pros and Cons of Point-Value Representation

- Computing the sum $A(x)+B(x)$ takes $\Theta(n)$,
- Computing the product $A(x)B(x)$ takes $\Theta(n)$ (need to have $2n$ points from each of A and B though)
- Inconvenient for user interaction

Problem

How to convert between the two representations efficiently?

- *Point-Value to Coefficient: interpolation problem*
- *Coefficient to Point-Value: evaluation problem*

Problem

How to multiply two polynomials in coefficient representation faster than $\Theta(n^2)$?

The Interpolation Problem

Given n point-value pairs $(x_i, A(x_i))$, find coefficients $a_0, \dots, a_{n-1}$

- Gaussian elimination helps solve it in $O(n^3)$ time.
- Lagrange's formula helps solve it in $\Theta(n^2)$ time:

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k}(x - x_j)}{\prod_{j \neq k}(x_k - x_j)}$$

- Fast Fourier Transform (FFT) helps perform the inverse DFT operation (another way to express interpolation) in $\Theta(n \lg n)$-time.

The Evaluation Problem

Given coefficients, evaluate $A(x_0), \ldots, A(x_{n-1})$

- Horner's rule gives $\Theta(n^2)$
- Again FFT helps perform the DFT operation in $\Theta(n \lg n)$-time

The Polynomial Multiplication Problem

Input: $A(x), B(x)$ of degree bound n in coefficient form
Output: $C(x) = A(x)B(x)$ of degree bound $2n - 1$ in coefficient form

1. Double degree bound: extend $A(x)$'s and $B(x)$'s coefficient representations to be of degree bound $2n$ [$\Theta(n)$]
2. Evaluate: compute point-value representations of $A(x)$ and $B(x)$ at each of the $2n$th roots of unity (with FFT of order $2n$) [$\Theta(n \lg n)$]
3. Point-wise multiply: compute point-value representation of $C(x) = A(x)B(x)$ [$\Theta(n)$]
4. Interpolate: compute coefficient representation of $C(x)$ (with FFT or order $2n$) [$\Theta(n \lg n)$]

Reminders on Complex Numbers

- $\mathbb{C} = \{a + bi \mid a, b \in \mathbb{R}\}$
- $w^n = 1 \Rightarrow w$ is a complex nth root of unity
- There are n of them: $\omega_n^k, k = 0, \ldots, n-1$, where $\omega_n = e^{2\pi i/n}$ is the principal nth root of unity
- In general, $\omega_n^j = \omega_n^{j \bmod n}$.

Lemma (Cancellation lemma)

$\omega_{dn}^{dk} = \omega_n^k$, $n \geq 0, k \geq 0$, *and* $d > 0$,

In particular, $\omega_{2m}^m = \omega_2 = -1$.

Lemma (Summation lemma)

Given $n \geq 1$, k *not divisible by* n, *then* $\sum_{j=0}^{n-1} (\omega_n^k)^j = 0$.

Discrete Fourier Transform (DFT)

Given $A(x) = \sum_{j=0}^{n-1} a_j x^j$, let $y_k = A(\omega_n^k)$, then the vector

$$\mathbf{y} = (y_0, y_1, \ldots, y_{n-1})$$

is the Discrete Fourier Transform (**DFT**) of the coefficient vector $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1})$.
We write

$$\mathbf{y} = \text{DFT}_n(\mathbf{a}).$$

Fast Fourier Transform

FFT is an efficient D&C **algorithm** to compute DFT (a transformation)
Idea: suppose $n = 2m$
1. Divide

$$\begin{aligned}
A(x) &= a_0 + a_1 x + a_2 x + \cdots + a_{2m-1}x^{2m-1} \\
&= a_0 + a_2 x^2 + a_4 x^4 + \cdots + a_{2m-2}x^{2m-2} + \\
&\quad x(a_1 + a_3 x^2 + a_5 x^4 + \cdots + a_{2m-1}x^{2m-2}) \\
&= A^{[0]}(x^2) + xA^{[1]}(x^2),
\end{aligned}$$

where

$$\begin{aligned}
A^{[0]}(x) &= a_0 + a_2 x + a_4 x^2 + \cdots + a_{2m-2}x^{m-1} \\
A^{[1]}(x) &= a_1 + a_3 x + a_5 x^2 + \cdots + a_{2m-1}x^{m-1}
\end{aligned}$$

FFT (continue)

By the cancellation lemma,

$$(\omega_{2m}^0)^2 = \omega_m^0, \quad (\omega_{2m}^1)^2 = \omega_m^1, \quad \dots, \quad (\omega_{2m}^{m-1})^2 = \omega_m^{m-1}$$

$$(\omega_{2m}^m)^2 = \omega_m^0, \quad (\omega_{2m}^{m+1})^2 = \omega_m^1, \quad \dots, \quad (\omega_{2m}^{2m-1})^2 = \omega_m^{m-1}$$

We thus get two smaller evaluation problems for $A^{[0]}(x)$ and $A^{[1]}(x)$:

$$\begin{aligned}
A(\omega_{2m}^j) &= A^{[0]}((\omega_{2m}^j)^2) + \omega_{2m}^j A^{[1]}((\omega_{2m}^j)^2) \\
&= A^{[0]}(\omega_m^j) + \omega_{2m}^j A^{[1]}(\omega_m^j) \\
&= A^{[0]}(\omega_m^{j \bmod m}) + \omega_{2m}^j A^{[1]}(\omega_m^{j \bmod m})
\end{aligned}$$

FFT (continue)

From $\mathbf{a} = (a_0, a_1, \ldots, a_{2m-1})$, we want $\mathbf{y} = \text{DFT}_{2m}(\mathbf{a})$.

2. Conquer

$$\mathbf{a}^{[0]} = (a_0, a_2, \ldots, a_{2m-2}), \quad \mathbf{a}^{[1]} = (a_1, a_3, \ldots, a_{2m-1})$$

$$\mathbf{y}^{[0]} = \text{DFT}_m(\mathbf{a}^{[0]}), \quad \mathbf{y}^{[1]} = \text{DFT}_m(\mathbf{a}^{[1]})$$

3. Combine $\mathbf{y}$ computed from $\mathbf{y}^{[0]}$ and $\mathbf{y}^{[1]}$ as follows.

For $0 \leq j \leq m-1$:

$$y_j = A(\omega_{2m}^j) = A^{[0]}(\omega_m^j) + \omega_{2m}^j A^{[1]}(\omega_m^j) = y_j^{[0]} + \omega_{2m}^j y_j^{[1]}.$$

For $m \leq j \leq 2m-1$:

$$y_j = A(\omega_{2m}^j) = A^{[0]}(\omega_m^{j-m}) + \omega_{2m}^j A^{[1]}(\omega_m^{j-m}) = y_{j-m}^{[0]} + \omega_{2m}^j y_{j-m}^{[1]} = y_{j-m}^{[0]} - \omega_{2m}^{j-m} y_{j-m}^{[1]}$$

FFT – Pseudo Code

RECURSIVE-FFT($\mathbf{a}$)

```
n ← length(a)   // n is a power of 2
if n = 1 then
    return a
end if
ω_n ← e^{2πi/n}   // principal nth root of unity
a^[0] ← (a_0, a_2, ..., a_{n-2}),  a^[1] ← (a_1, a_3, ..., a_{n-1})
y^[0] ← RECURSIVE-FFT(a^[0]),  y^[1] ← RECURSIVE-FFT(a^[1])
w ← 1   really meant w ← ω_n^0
for k ← 0 to n/2 − 1 do
    y_k ← y_k^[0] + w y_k^[1],   y_{k+n/2} ← y_k^[0] − w y_k^[1]
    w ← w ω_n
end for
return y
```

$$T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \lg n)$$

Inverse DFT – Interpolation at the Roots

Now that we know $\mathbf{y}$, how to compute $\mathbf{a} = \text{DFT}_n^{-1}(\mathbf{y})$?

$$\begin{bmatrix} 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \dots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

Need the inverse V_n^{-1} of $V_n := V(1, \omega_n, \omega_n^2 \dots, \omega_n^{n-1})$

Inverse DFT – Interpolation at the Roots

Theorem

For $0 \le j, k \le n-1$,

$$[V_n^{-1}]_{j,k} = \frac{\omega_n^{-kj}}{n}.$$

Thus,

$$a_j = \sum_{k=0}^{n-1} [V_n^{-1}]_{j,k} y_k = \sum_{k=0}^{n-1} \frac{\omega_n^{-kj}}{n} y_k = \sum_{k=0}^{n-1} \frac{y_k}{n} (\omega_n^{-j})^k$$

$$a_j = Y(\omega_n^{-j}), \quad Y(x) = \frac{y_0}{n} + \frac{y_1}{n} x + \dots + \frac{y_{n-1}}{n} x^{n-1}$$

We can easily modify the pseudo code for FFT to compute $\mathbf{a}$ from $\mathbf{y}$ in $\Theta(n \lg n)$-time (homework!)