南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

05 Divide and Conquer

CS216 Algorithm Design and Analysis (H)

Instructor: Shan Chen

chens3@sustech.edu.cn

Divide-and-Conquer Paradigm

- **Divide and conquer:**
  - Divide up problem into several subproblems (of the same kind).
  - Conquer (solve) each subproblem recursively.
  - Combine solutions to subproblems into solution to original problem.
- **Most common usage:**
  - Divide problem of size n into two subproblems of size $n/2$.
  - Conquer (solve) two subproblems recursively.
  - Combine two solutions into solution to original problem.

e.g., combine in O(n) time

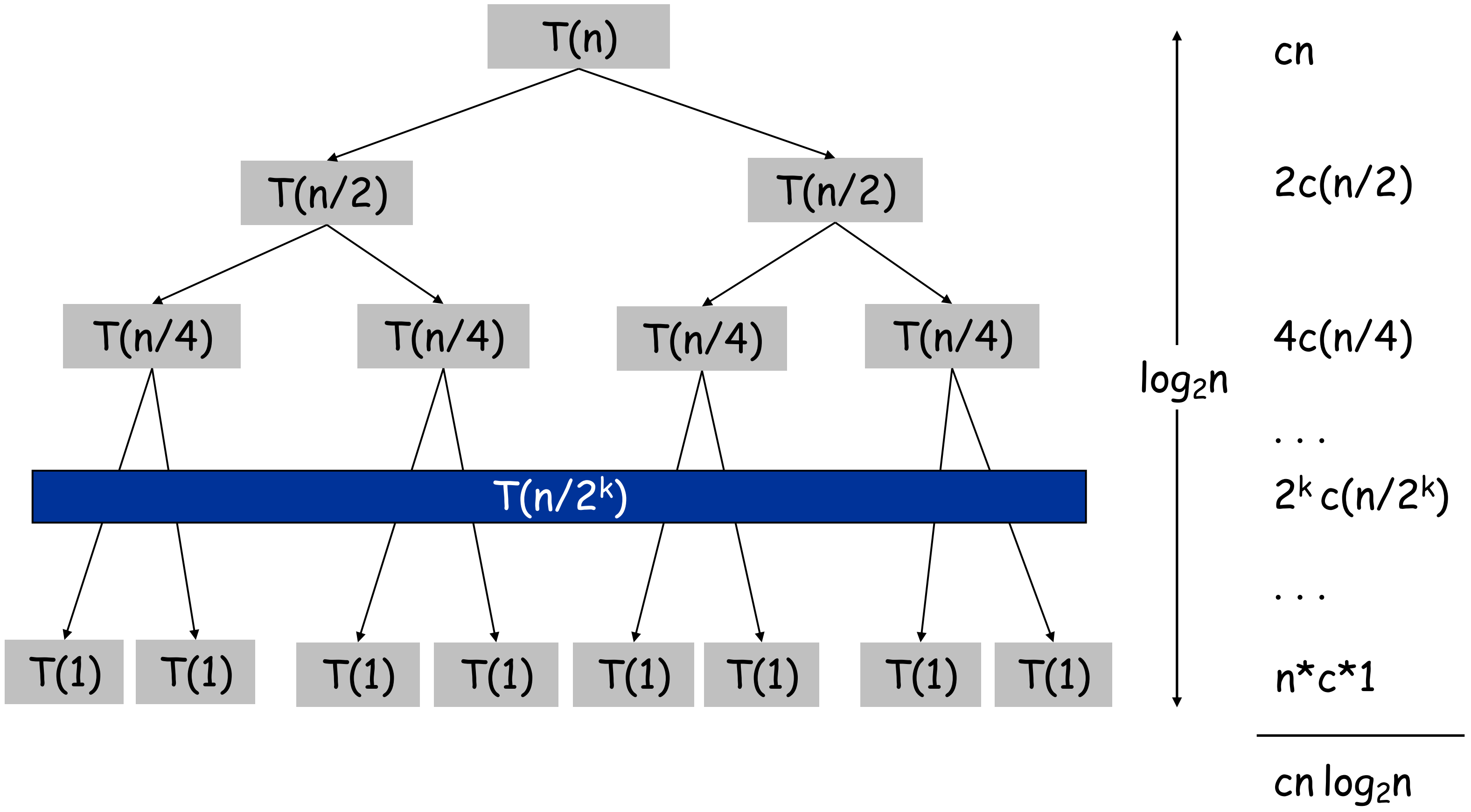
- **Common time complexity:**
  - Brute force: $\Theta(n^2)$
  - Divide and conquer: $O(n \log n)$ ← $T(n) = 2T(n/2) + O(n)$

凡治众如治寡
分数是也
孙子兵法

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Divide-and-Conquer Recurrences

- **Example:** $T(n) = 2T(n/2) + O(n)$ with $T(1) = \Theta(1)$



Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Divide-and-Conquer Recurrences

- **Divide-and-conquer recurrences.** $T(n) = aT(n/b) + f(n)$ with $T(1) = \Theta(1)$

- **Master theorem.** Let $a \geq 1, b \geq 2, c \geq 0$ and suppose $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) = aT(n/b) + \Theta(n^c)$$

with $T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

  - Case 1: If $c > \log_b a$, then $T(n) = \Theta(n^c)$.
  - Case 2: If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.
  - Case 3: If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

PEARSON Addison Wesley
Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

1. Counting Inversions

Counting Inversions

- **Match song preferences.**
  - Every person ranks n songs.
  - Music site consults database to find people with similar tastes.
- **Similarity metric.** Number of inversions between two rankings.
  - My rank: $1, 2, \ldots, n$
  - Your rank: $a_1, a_2, \ldots, a_n$
  - Songs i and j are inverted if $i < j$ but $a_i > a_j$.

Songs

	A	B	C	D	E
Me	1	2	3	4	5
You	1	3	4	2	5

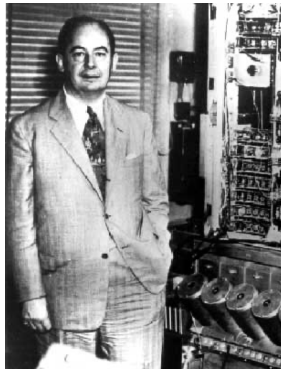
inversions
3-2, 4-2

- **Brute force.** Check all $\Theta(n^2)$ pairs.
- **Q.** Can we count inversions faster?

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Recall: Mergesort

- **Mergesort:**
  - Divide array into two halves.
  - Recursively sort each half.
  - Merge two halves to make sorted whole.



Jon von Neumann (1945)

A	L	G	O	R	I	T	H	M	S

A	L	G	O	R	I	T	H	M	S

divide $O(n)$

A	G	L	O	R	H	I	M	S	T

sort $2T(n/2)$

A	G	H	I	L	M	O	R	S	T

merge $O(n)$

total $O(n \log n)$

PEARSON Addison Wesley
Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Counting Inversions: Divide and Conquer

- **Divide and conquer:**
  - Divide: separate list into two pieces.
  - Conquer: recursively count inversions in each half.
  - Combine: count inversions where a_i and a_j are in different halves and return sum of three quantities.

1	5	4	8	10	2	6	9	12	11	3	7

Divide: O(n)

1	5	4	8	10	2

6	9	12	11	3	7

Conquer: 2T(n/2)

5 blue-blue inversions

5-4, 5-2, 4-2, 8-2, 10-2

8 green-green inversions

6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7

9 blue-green inversions

5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

Count: O(n) ?

how to count blue-green inversions ?

Total = 5 + 8 + 9 = 22.

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Counting Inversions: Divide and Conquer

- **Divide and conquer:**
  - Divide: separate list into two pieces.
  - Conquer: recursively count inversions in each half.
  - Combine: count inversions where a_i and a_j are in different halves and return sum of three quantities.

easy to count if both sorted

1	5	4	8	10	2	6	9	12	11	3	7

Divide: O(n)

1	2	4	5	8	10

3	6	7	9	11	12
4	2	2	1	0	0

Conquer: 2T(n/2)

blue-green inversions: 4 + 2 + 2 + 1 + 0 + 0 = 9

Count: O(n)

Merge-and-Count: O(n)

1	2	3	4	5	6	7	8	9	10	11	12

Merge: O(n)

PEARSON Addison Wesley
Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Counting Inversions: *O(n log n)* Algorithm

- **Sort-and-Count algorithm:**

```
Sort-and-Count(L) {
   if (list L has one element) return (0, L)

   Divide the list into two halves A and B
      (rA, A) ← Sort-and-Count(A)
      (rB, B) ← Sort-and-Count(B)
      (rAB, L) ← Merge-and-Count(A, B)

   return (rA + rB + rAB, L)
}
```

- **Merge-and-Count sub-algorithm:**
  - Input: *A* and *B* are sorted (by Sort-and-Count).
  - Output: *L* is sorted.

PEARSON Addison Wesley
Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

2. Closest Pair of Points

Closest Pair of Points

- **Closest pair problem.** Given *n* points in the plane, find a pair with smallest Euclidean distance between them.
- **Fundamental geometric primitive.**
  - Example applications: graphics, computer vision, geographic information systems, molecular modeling, air traffic control, etc.
  - Special case of nearest neighbor, Euclidean MST, Voronoi diagram, etc.

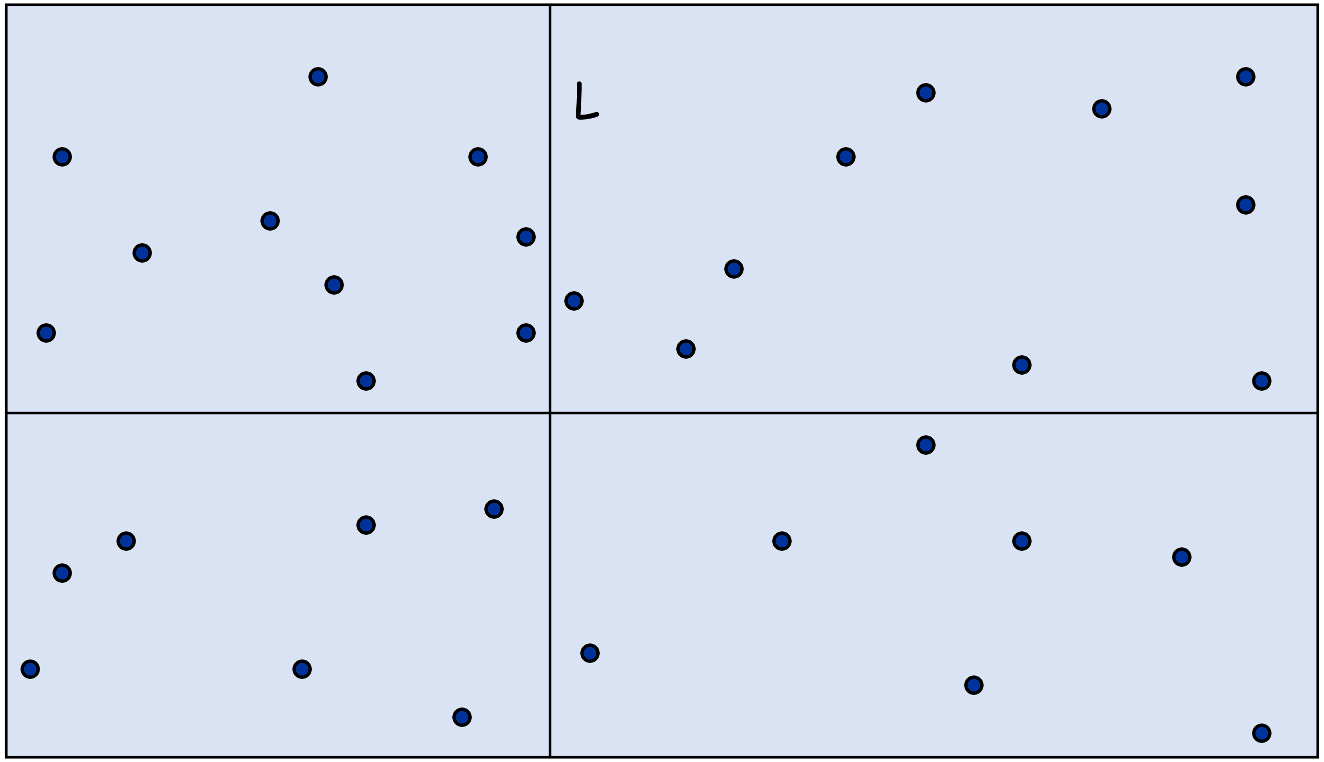
fast closest pair inspired fast algorithms for these problems

- **Brute force.** Check all pairs with $\Theta(n^2)$ distance calculations.
- **1-D version.** Easy $O(n \log n)$ algorithm if points are on a line.

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Closest Pair of Points: A First Attempt

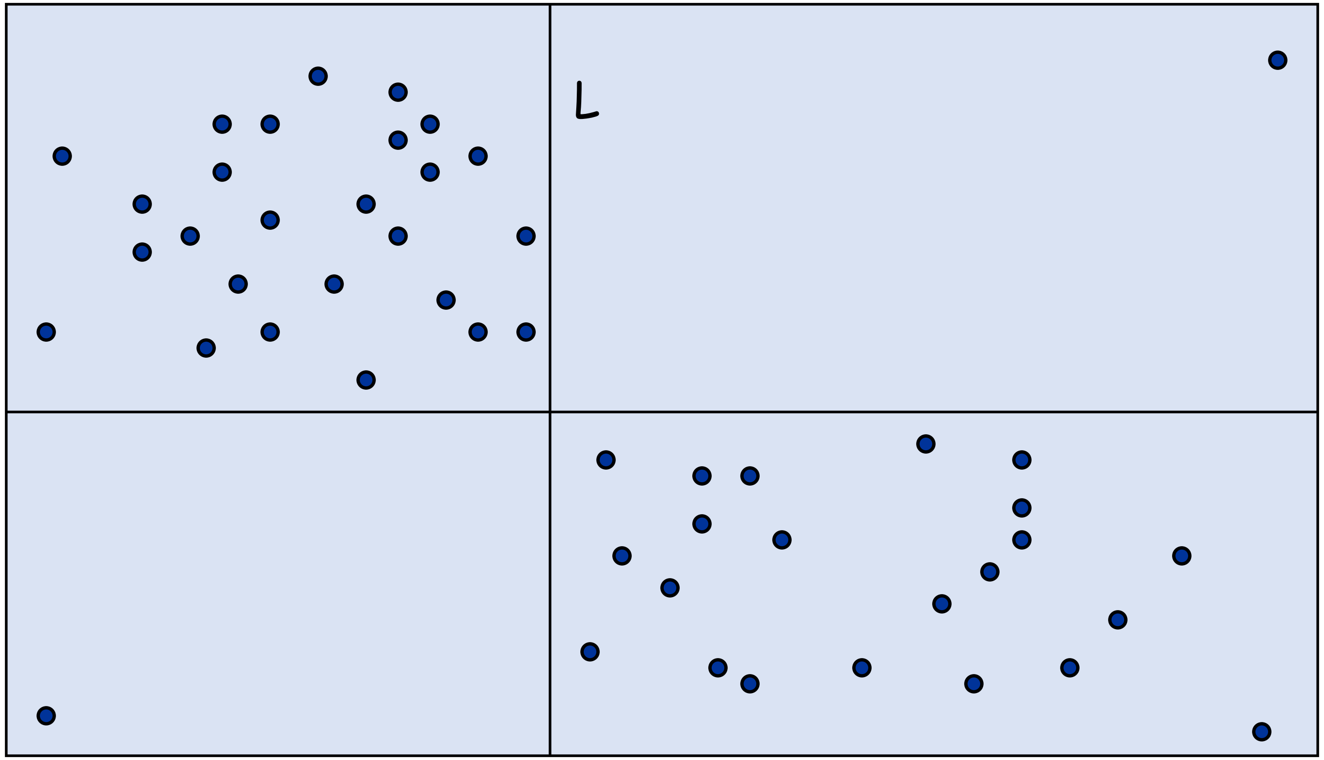
- **Divide.** Sub-divide region into *4* quadrants.



Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Closest Pair of Points: A First Attempt

- **Divide.** Sub-divide region into *4* quadrants.
- **Obstacle.** Impossible to ensure *n/4* points in each quadrant.

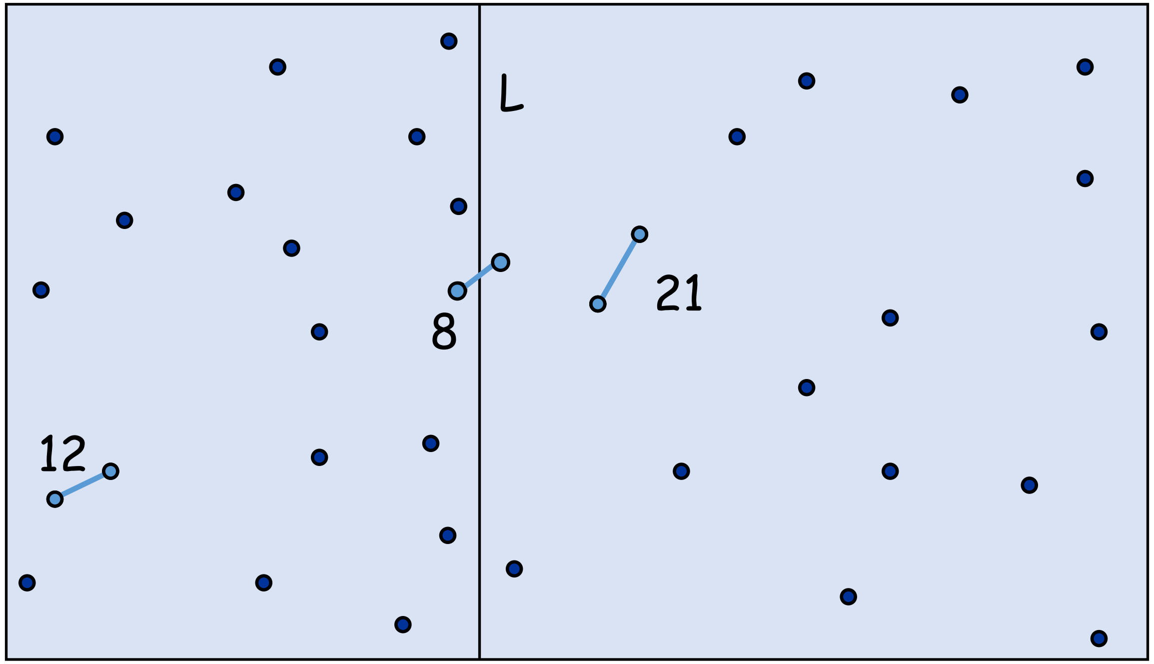


Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Closest Pair of Points: Divide and Conquer

- **Divide and conquer:**
  - Divide: draw vertical line *L* so that roughly *n/2* points lie in each side.
  - Conquer: find closest pair in each side recursively.
  - Combine: find closest pair with one point in each side and return best of *3* solutions.

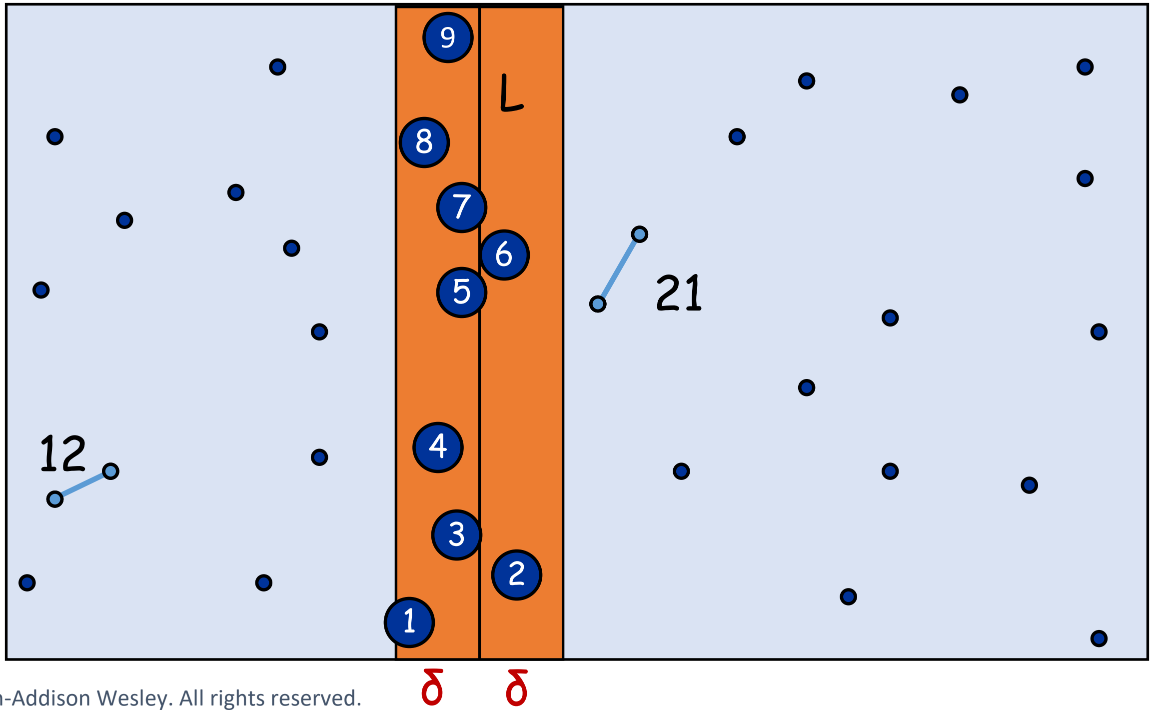
can we beat $\Theta(n^2)$?



Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Closest Pair of Points: Divide and Conquer

- **Combine step.** Find closest pair with one point in each side.
  - Observation: suffices to consider only points within δ of line L, where δ is the distance of closest pair with both points in one side.
    known from recursion
  - Sort points in 2δ-strip by their y-coordinate.
  - Check distances of only those points in the sorted list within 7 positions!
    why this works?



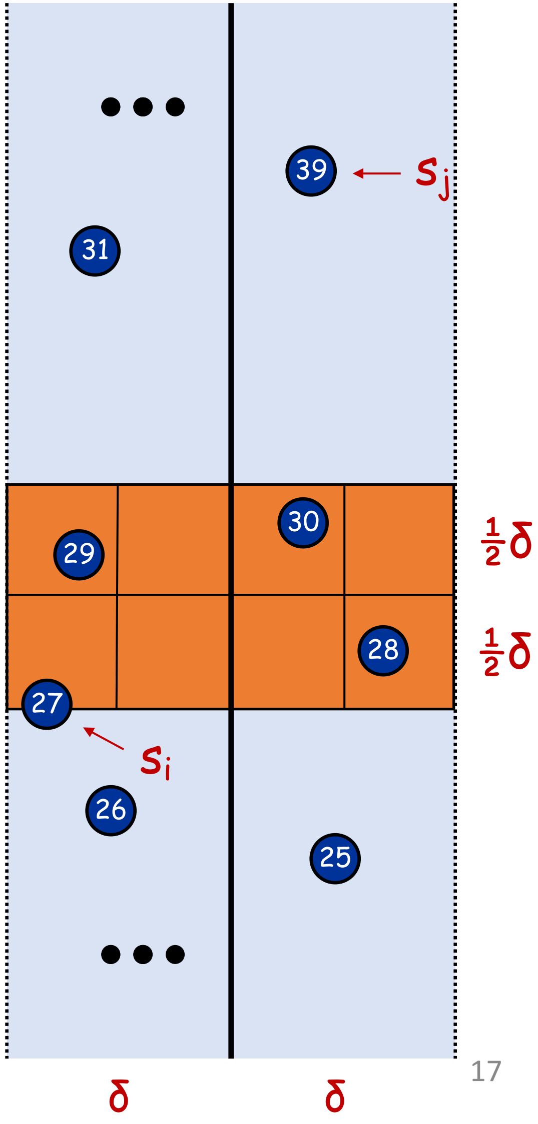
δ = min (12, 21)

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Closest Pair of Points: Divide and Conquer

- **Claim.** Let s_i be the point in the 2δ-strip with the i-th smallest y-coordinate. If $|i - j| > 7$, then the distance between s_i and s_j is at least δ.
- **Pf.** (direct proof)
  - Consider the 2δ-by-δ rectangle R in strip whose min y-coordinate is y-coordinate of s_i.
  - Distance between s_i and any point s_j above R is $\geq \delta$.
  - Subdivide R into 8 squares.
  - At most 1 point per square. ← because square diameter < δ
  - At most 7 other points can be in R. ▪

constant can be improved with more refined argument



Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Closest Pair of Points: *O(n log n)* Algorithm

- **Divide-and-conquer algorithm:** [sort by x-axis and y-axis beforehand]

```
Closest-Pair(p1, ..., pn) {
   Compute vertical line L such that half the points
   are on each side of the line and Partition the points.

   δ1 ← Closest-Pair(left half)
   δ2 ← Closest-Pair(right half)
   δ  ← min(δ1, δ2)

   Delete all points further than δ from line L.

   Sort remaining points by y-coordinate.

   Scan points in y-order and compare distance between
   each point and next 7 neighbors. If any of these
   distances is less than δ, update δ.

   return δ
}
```

$O(n)$ use x-sorted list

$2T(n / 2)$

$O(n)$ use y-sorted list

~~O(n log n)~~ $O(n)$

$O(n)$

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Closest Pair of Points: Closing Remarks

- [Rabin 1976] There exists a randomized algorithm that finds the closest pair of points in the plane with expected running time $O(n)$.

- **Remark.** There are ingenious divide-and-conquer algorithms for core geometric problems.
  - 3D or higher dimensions test limits of our ingenuity.

problem	brute	clever
closest pair	$O(n^2)$	$O(n \log n)$
farthest pair	$O(n^2)$	$O(n \log n)$
convex hull	$O(n^2)$	$O(n \log n)$
Delaunay/Voronoi	$O(n^4)$	$O(n \log n)$
Euclidean MST	$O(n^2)$	$O(n \log n)$

running time to solve a 2D problem with n points

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

3. Median and Selection

An example of randomized algorithms

[section 13.5 of textbook]

Median and Selection

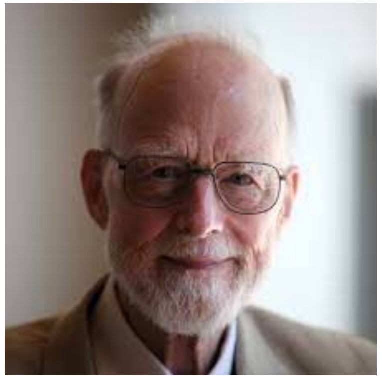
- **Median and selection.** Given n elements from a totally ordered universe, find the median element or in general the k-th smallest element.
  - minimum or maximum ($k = 1$ or $k = n$): $O(n)$ compares
  - median: $k = \lfloor (n + 1) / 2 \rfloor$
    - ✓ $O(n \log n)$ compares by sorting
    - ✓ $O(n \log k)$ compares with a binary heap
- **Applications.** Order statistics, find the "top k", bottleneck paths, etc.

- **Q.** Can we do it with $O(n)$ compares?
- **A.** Yes! Selection is easier than sorting.

PEARSON Addison Wesley
Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Recall: Randomized Quicksort

- **Randomized quicksort:**
  - Pick a random pivot element p.
  - 3-way partition the array into L, M, and R.
    - ✓ L: elements $< p$, M: elements $= p$, R: elements $> p$.
  - Recursively sort both L and R.



Tony Hoare (1959)

A	L	G	O	R	I	T	H	M	S

pick random pivot $O(1)$

A	L	G	I	H	M	O	R	T	S

3-way partition $O(n)$

A	G	H	I	L	M	O	R	S	T

sort $T(|L|) + T(|R|)$

A	G	H	I	L	M	O	R	S	T

total $O(n \log n)$ on average

PEARSON Addison Wesley

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Median and Selection: Divide-and-Conquer

- **Divide and conquer:**
  - Pick a random pivot element p.
  - 3-way partition the array into L, M, and R.
    - ✓ L: elements $< p$, M: elements $= p$, R: elements $> p$.
  - Recursively select in one subarray: the one containing the k-th smallest element.

A	L	G	O	R	I	T	H	M	S

pick random pivot $O(1)$

A	L	G	I	H	M	O	R	T	S

3-way partition $O(n)$

select $T(|L|)$ or $O(1)$ or $T(|R|)$

PEARSON Addison Wesley
Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Randomized Quickselect

- **Randomized quickselect.** Divide and select.

```
Quick-Select(A, k) { // 1 ≤ k ≤ |A|
   Pick pivot p uniformly at random from A
   Partition the list into two three parts L, M and R

   if (k ≤ |L|)
      return Quick-Select(L, k)
   else if (k > |L| + |M|)
      return Quick-Select(R, k - |L| - |M|)
   else
      return p
}
```

- **Q.** What is the expected time complexity of randomized quickselect?
   - ➢ Time complexity is measured by the number of compares.

PEARSON Addison Wesley
Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Randomized Quickselect: Time Complexity

- **Intuition.** Split a length-n array uniformly $\Rightarrow$ expected larger size $\sim 3n/4$.
  - $T(n) \le T(3n/4) + n \Rightarrow T(n) \le 4n$

    not rigorous: cannot assume $E[T(i)] \le T(E[i])$

- **Def.** Let $T(n, k)$ be the expected number of compares to select the k-th smallest element in an array of length n. Let $T(n) = \max_k T(n, k)$.
- **Claim.** $T(n) \le 4n$
- **Pf.** (by strong induction on n)

T(i) ≤ T(n - i) since T(n) is monotonely non-decreasing

$$
\begin{aligned}
T(n) &\le n + 1/n\, [\, 2T(n/2) + \ldots + 2T(n-3) + 2T(n-2) + 2T(n-1)\,] \\
&\le n + 1/n\, [\, 8(n/2) + \ldots + 8(n-3) + 8(n-2) + 8(n-1)\,] \\
&\le n + 1/n\, (3n^2) \\
&= 4n
\end{aligned}
$$

inductive hypothesis

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Median and Selection: Closing Remarks

- We learned that randomized quickselect runs in $O(n)$ time on average.

- [Blum-Floyd-Pratt-Rivest-Tarjan 1973] There exists a compare-based deterministic selection algorithm whose worst-case running time is $O(n)$.
  - This algorithm is also known as median-of-medians selection.
  - Optimized version requires $\leq 5.4305n$ compares.

- **Remark.** In practice, we use randomized selection algorithms since deterministic algorithms have too large constants.
  - However, deterministic algorithms can be used as a fallback for pivot selection.

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

4. Integer and Matrix Multiplication

Integer Addition and Subtraction

- **Addition.** Given two n-bit integers a and b, compute $a + b$.
- **Subtraction.** Given two n-bit integers a and b, compute $a - b$.
- **Grade-school addition/subtraction algorithm.** $\Theta(n)$ bit operations.

1	1	1	1	1	1	0	1	
	1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	0	1
1	0	1	0	1	0	0	1	0

- **Remark.** Grade-school addition and subtraction algorithms are optimal.

PEARSON Addison Wesley

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Integer Multiplication

- **Multiplication.** Given two n-bit integers a and b, compute $a \times b$.
- **Grade-school multiplication algorithm.** $\Theta(n^2)$ bit operations.

- **Conjecture.** [Kolmogorov 1956]
  Grade-school multiplication algorithm is optimal.
- **Theorem.** [Karatsuba 1960] Conjecture is false.

```
                  1 1 0 1 0 1 0 1
                × 0 1 1 1 1 1 0 1
                  1 1 0 1 0 1 0 1
                0 0 0 0 0 0 0 0 0
              1 1 0 1 0 1 0 1 0
            1 1 0 1 0 1 0 1 0
          1 1 0 1 0 1 0 1 0
        1 1 0 1 0 1 0 1 0
      1 1 0 1 0 1 0 1 0
    0 0 0 0 0 0 0 0 0
  0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1
```


PEARSON Addison Wesley
Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Integer Multiplication: A First Attempt

- **Divide and conquer:** (multiply two n-bit integers x and y)
  - Divide x and y into low- and high-order bits.
  - Recursively multiply four $n/2$-bit integers: ac, bc, ad, bd
  - Add and shift to obtain result.
- **Example.** $n = 8, m = \lceil n/2 \rceil = 4.$

$$x = \underbrace{1000}_{a}\underbrace{1101}_{b} \qquad y = \underbrace{1110}_{c}\underbrace{0001}_{d}$$

$$xy = (2^m a + b)(2^m c + d) = 2^{2m}\underbrace{ac}_{1} + 2^m(\underbrace{bc}_{2} + \underbrace{ad}_{3}) + \underbrace{bd}_{4}$$

- **Time complexity.** $\Theta(n^2)$ $\longleftarrow$ $T(n) = 4T(n/2) + O(n)$

PEARSON Addison Wesley
Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Integer Multiplication: Karatsuba's Trick

- **Divide and conquer:** (multiply two n-bit integers x and y)
  - Divide x and y into low- and high-order bits.
  - Recursively multiply three $n/2$-bit integers: ac, ~~bc, ad,~~ $(a - b)\ (c - d)$, bd
  - Add and shift to obtain result.
- **Example.** $n = 8$, $m = \lceil n/2 \rceil = 4$.

$$x = \underbrace{1000}_{a}\underbrace{1101}_{b} \qquad y = \underbrace{1110}_{c}\underbrace{0001}_{d}$$

bc + ad = ac + bd - (a - b)(c - d)

$$xy = (2^m a + b)(2^m c + d) = 2^{2m}\underset{1}{ac} + 2^m(\underset{1}{ac} + \underset{3}{bd} - \underset{2}{(a - b)\ (c - d)}) + \underset{3}{bd}$$

- **Time complexity.** $\Theta(n^{\log_2 3}) = \Theta(n^{1.585})$ ⟵ T(n) = 3T(n/2) + O(n)

PEARSON Addison Wesley

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Karatsuba's Algorithm

- **Karatsuba's algorithm:**

```
Karatsuba-Multiply(x, y, n) {
   if (n = 1) return x * y
   else
      m ← ⌈ n / 2 ⌉
      a ← ⌊ x / 2^m ⌋; b = x mod 2^m
      c ← ⌊ y / 2^m ⌋; d = y mod 2^m
      e ← Karatsuba-Multiply(a, c, m)
      f ← Karatsuba-Multiply(b, d, m)
      g ← Karatsuba-Multiply(|a – b|, |c – d|, m)
      Flip sign of g if needed

      return 2^2m e + 2^m(e + f – g) + f
}
```

$O(n)$

$3T(n / 2)$

$O(n)$

- **Practice.** Use *32/64*-bit word. Faster than grade-school for ≥~*320* bits.

PEARSON Addison Wesley
Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Integer Multiplication: Asymptotic Complexity

year	algorithm	bit operations
12xx	grade school	$O(n^2)$
1962	Karatsuba–Ofman	$O(n^{1.585})$
1963	Toom–3, Toom–4	$O(n^{1.465})$, $O(n^{1.404})$
1966	Toom–Cook	$O(n^{1+\varepsilon})$
1971	Schönhage–Strassen	$O(n \log n \cdot \log \log n)$
2007	Fürer	$n \log n \, 2^{O(\log^* n)}$
2019	Harvey–van der Hoeven	$O(n \log n)$
	???	$O(n)$

- **Remark.** GNU Multiple Precision Arithmetic Library (GMP) uses one of first five algorithms depending on n.

used in Maple, Mathematica, gcc, cryptography…

PEARSON Addison Wesley

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Matrix Multiplication

- **Matrix multiplication.** Given *n*-by-*n* matrices *A* and *B*, compute *C = AB*.
- **Grade-school matrix multiplication.** $\Theta(n^3)$ arithmetic operations.
- **Block matrix multiplication:**

$$\begin{bmatrix} 152 & 158 & 164 & 170 \\ 504 & 526 & 548 & 570 \\ 856 & 894 & 932 & 970 \\ 1208 & 1262 & 1316 & 1370 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix} \times \begin{bmatrix} 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 \\ 28 & 29 & 30 & 31 \end{bmatrix}$$

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$$

PEARSON Addison Wesley

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Matrix Multiplication: A First Attempt

- **Divide and conquer:** (multiply two *n*-by-*n* matrices *A* and *B*)
  - Divide: partition *A* and *B* into *n/2*-by-*n/2* blocks.
  - Conquer: multiply *8* pairs of *n/2*-by-*n/2* matrices, recursively.
  - Combine: add appropriate products using *4* matrix additions.

n-by-n matrices

$$C = A \times B$$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

½n-by-½n matrices

8 matrix multiplications
(of ½n-by-½n matrices)

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

4 matrix additions
(of ½n-by-½n matrices)

$T(n) = 8T(n/2) + O(n^2)$

$T(n) = O(n^3)$

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Matrix Multiplication: Strassen's Trick

- **Divide and conquer:** (multiply two *n*-by-*n* matrices *A* and *B*)
  - Divide: partition *A* and *B* into *n/2*-by-*n/2* blocks.
  - Conquer: multiply *7* pairs of *n/2*-by-*n/2* matrices, recursively.
  - Combine: *11* matrix additions and *7* matrix subtractions.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

scalars

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_1 + P_5 - P_3 - P_7 \end{aligned}$$

$$\begin{aligned} P_1 &\leftarrow A_{11} \times (B_{12} - B_{22}) \\ P_2 &\leftarrow (A_{11} + A_{12}) \times B_{22} \\ P_3 &\leftarrow (A_{21} + A_{22}) \times B_{11} \\ P_4 &\leftarrow A_{22} \times (B_{21} - B_{11}) \\ P_5 &\leftarrow (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &\leftarrow (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &\leftarrow (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

$T(n) = 7T(n/2) + O(n^2)$

$T(n) = O(n^{\log_2 7}) = O(n^{2.81})$

PEARSON Addison Wesley

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Strassen's Algorithm in Practice

- **Implementation issues:**
  - Sparsity.
  - Caching.
  - *n* not a power of *2*.
  - Numerical stability.
  - Non-square matrices.
  - Storage for intermediate submatrices.
  - Crossover to classical algorithm when *n* is "small".
  - Parallelism for multi-core and many-core architectures.
- **Nevertheless, still useful in practice.**
  - Apple reports *8x* speedup when $n \approx 2{,}048$.

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Matrix Multiplication: Asymptotic Complexity

Year	Bound on omega	Authors
1969	2.8074	Strassen[1]
1978	2.796	Pan[10]
1979	2.780	Bini, Capovani [it], Romani[11]
1981	2.522	Schönhage[12]
1981	2.517	Romani[13]
1981	2.496	Coppersmith, Winograd[14]
1986	2.479	Strassen[15]
1990	2.3755	Coppersmith, Winograd[16]
2010	2.3737	Stothers[17]
2012	2.3729	Williams[18][19]
2014	2.3728639	Le Gall[20]
2020	2.3728596	Alman, Williams[21][22]
2022	2.371866	Duan, Wu, Zhou[23]
2024	2.371552	Williams, Xu, Xu, and Zhou[2]

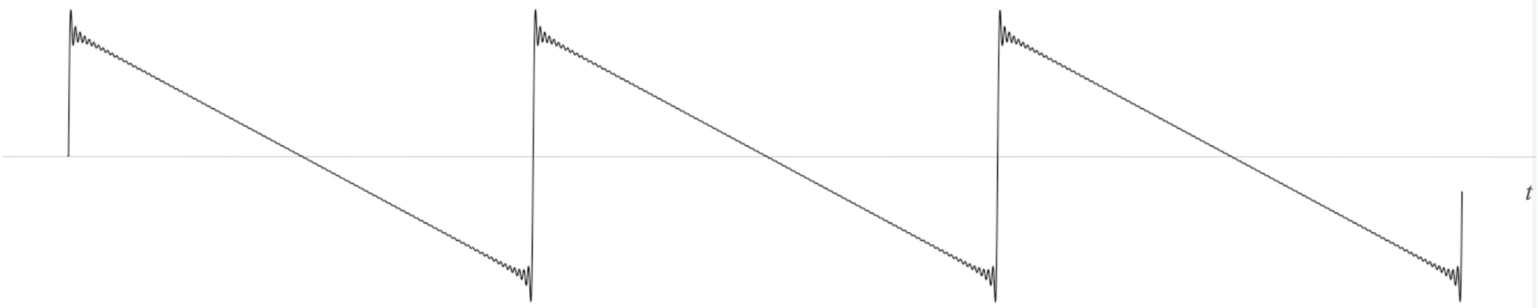
Reference: https://en.wikipedia.org/wiki/Computational_complexity_of_matrix_multiplication

SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

5. Convolution and FFT

Fourier Analysis and Euler's Identity

- **Fourier theorem.** [Fourier, Dirichlet, Riemann] Any (sufficiently smooth) periodic function can be expressed as the sum of a series of sinusoids.



$$y(t) = \frac{2}{\pi} \sum_{k=1}^{n} \frac{\sin kt}{k} \qquad n = 100$$

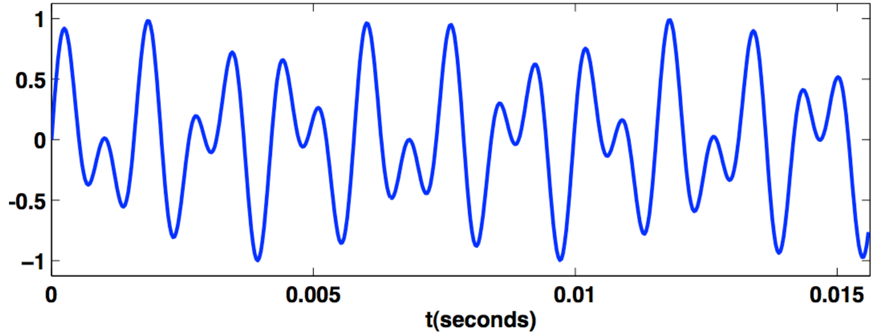
- **Euler's identity.** $e^{ix} = \cos x + i \sin x$
  - ➢ Sum of sine and cosines = sum of complex exponentials

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

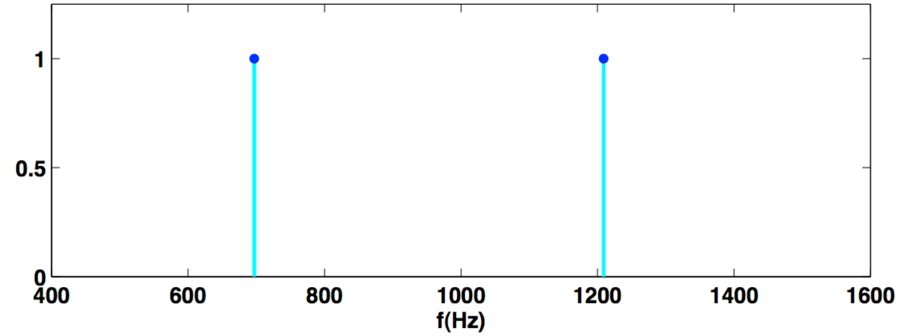
Example: Touch Tone

- **Signal for button 1.** $y(t) = \frac{1}{2}\sin(2\pi \cdot 697t) + \frac{1}{2}\sin(2\pi \cdot 1209t)$

- **Time domain:**



- **Frequency domain:**



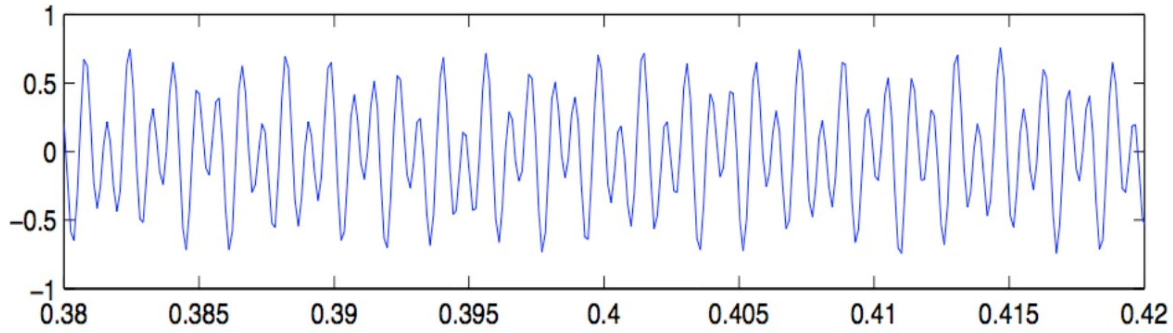
Reference: Cleve Moler, Numerical Computing with MATLAB

PEARSON Addison Wesley
Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

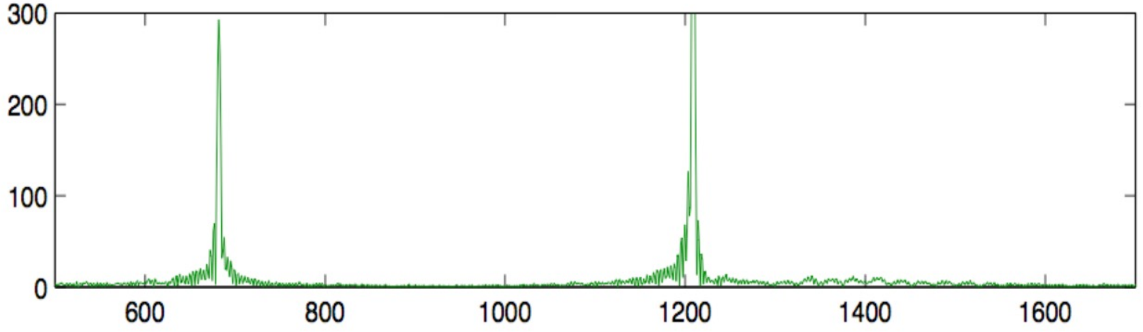
Example: Touch Tone

- **Signal for button 1.** $y(t) = \frac{1}{2} \sin(2\pi \cdot 697t) + \frac{1}{2} \sin(2\pi \cdot 1209t)$

- **Signal:**
  - sample rate: *~8kHz*



- **Magnitude of discrete Fourier transform:**

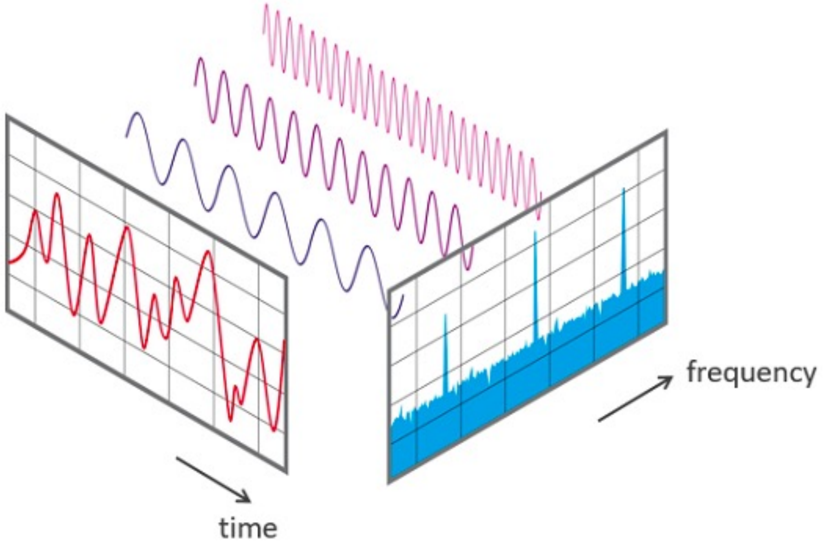


Reference: Cleve Moler, Numerical Computing with MATLAB

PEARSON Addison Wesley
Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

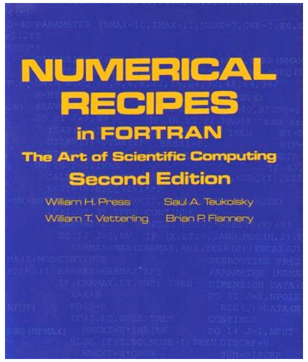
Fast Fourier Transform (FFT)

- **FFT.** Fast way to convert between time domain and frequency domain.



we take this viewpoint

- **Alternative viewpoint.** Fast way to multiply and evaluate polynomials.



" If you speed up any nontrivial algorithm by a factor of a million or so the world will beat a path towards finding useful applications for it. "

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Fast Fourier Transform: Applications

- **Applications:**
  - Optics, acoustics, quantum physics, telecommunications, radar, control systems, signal processing, speech recognition, data compression, image processing, seismology, mass spectrometry, ...
  - Digital media. [DVD, JPEG, MP3, H.264]
  - Medical diagnostics. [MRI, CT, PET scans, ultrasound]
  - Numerical solutions to Poisson's equation.
  - Integer and polynomial multiplication.
  - Shor's quantum factoring algorithm.
  - ....

> The FFT is one of the truly great computational developments of this [20th] century. It has changed the face of science and engineering so much that it is not an exaggeration to say that life as we know it would be very different without the FFT. --- Charles van Loan

PEARSON Addison Wesley

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

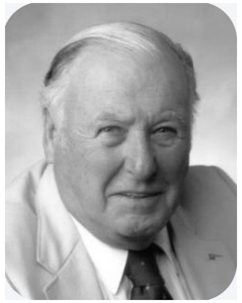
Fast Fourier Transform: Brief History

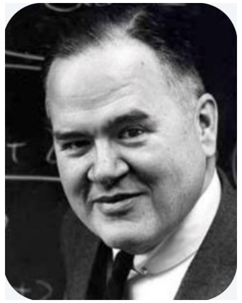
- [Gauss 1805, 1866] Analyzed periodic motion of asteroid Ceres.
- [Runge-König 1924] Laid theoretical groundwork.
- [Danielson-Lanczos 1942] Efficient algorithm, *x*-ray crystallography.
- [Cooley-Tukey 1965] Detect nuclear tests in Soviet Union and track submarines. Rediscovered and popularized FFT.

An Algorithm for the Machine Calculation of Complex Fourier Series

By James W. Cooley and John W. Tukey

An efficient method for the calculation of the interactions of a 2^m factorial experiment was introduced by Yates and is widely known by his name. The generalization to 3^m was given by Box et al. [1]. Good [2] generalized these methods and gave elegant algorithms for which one class of applications is the calculation of Fourier series. In their full generality, Good's methods are applicable to certain problems in which one must multiply an N-vector by an $N \times N$ matrix which can be factored into m sparse matrices, where m is proportional to $\log N$. This results in a procedure requiring a number of operations proportional to $N \log N$ rather than N^2.





- **Note.** Importance not fully realized until emergence of digital computers.

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Polynomials: Coefficient Representation

- **Univariate polynomial.** [coefficient representation]

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} \qquad B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1}$$

- **Addition.** *O(n)* arithmetic operations.

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

- **Evaluation.** *O(n)* using Horner's method.

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

- **Multiplication** (linear convolution): *O(n²)* using brute force.

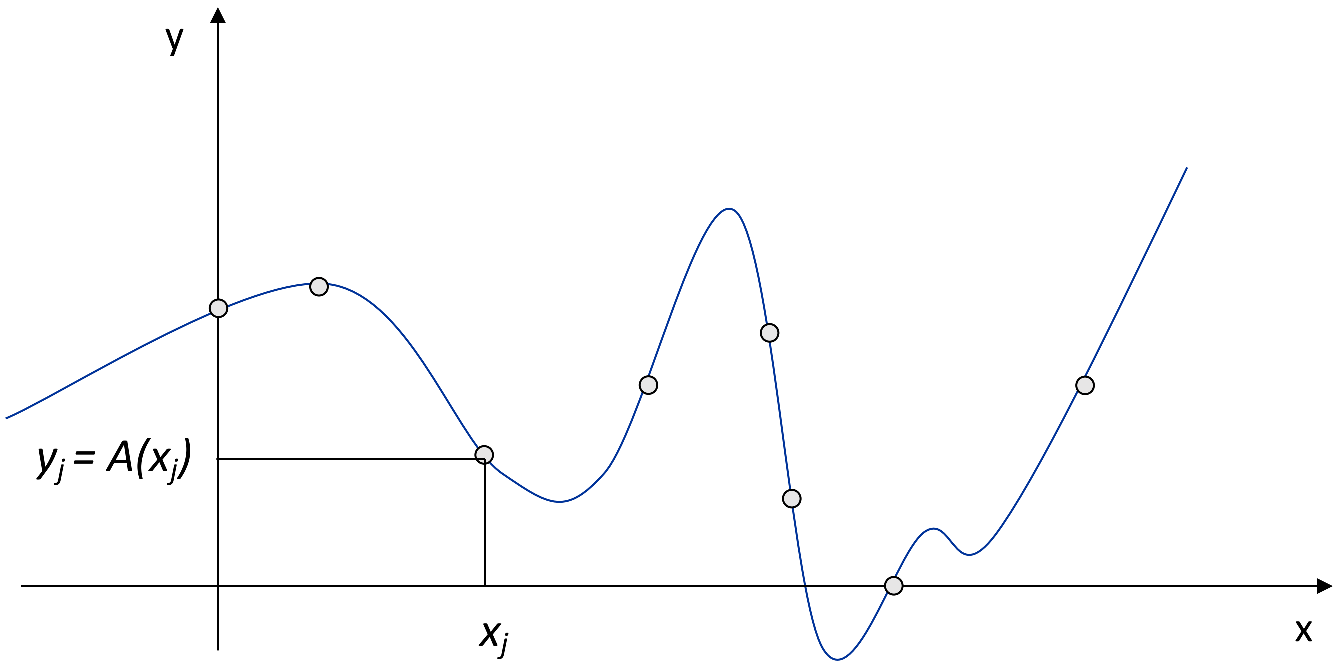
$$\boxed{A(x) \times B(x) = \sum_{i=0}^{2n-2} c_i x^i, \text{ where } c_i = \sum_{j=0}^{i} a_j b_{i-j}}$$

PEARSON Addison Wesley

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Polynomials: Point-Value Representation

- **Fundamental theorem of algebra.** [Gauss, PhD thesis] A degree n polynomial with complex coefficients has n complex roots.
- **Corollary.** A degree $n - 1$ polynomial $A(x)$ is uniquely specified by its evaluation at n distinct values of x.



Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Polynomials: Point-Value Representation

- **Polynomial.** [point-value representation]

$$A(x): \ (x_0, y_0), \ldots, (x_{n-1}, y_{n-1}) \qquad B(x): \ (x_0, z_0), \ldots, (x_{n-1}, z_{n-1})$$

- **Addition.** *O(n)* arithmetic operations.

$$A(x)+B(x): \ (x_0, y_0+z_0), \ldots, (x_{n-1}, y_{n-1}+z_{n-1})$$

- **Multiplication.** *O(n)*, but represent *A(x)* and *B(x)* using 2*n* points.

$$A(x) \times B(x): \ (x_0, y_0 \times z_0), \ldots, (x_{2n-1}, y_{2n-1} \times z_{2n-1})$$

- **Evaluation.** *O(n²)* using Lagrange's method.

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)} \quad \longleftarrow \text{not used in FFT}$$

PEARSON Addison Wesley

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Converting between Two Representations

- **Tradeoff.** Either fast evaluation or fast multiplication. We want both!

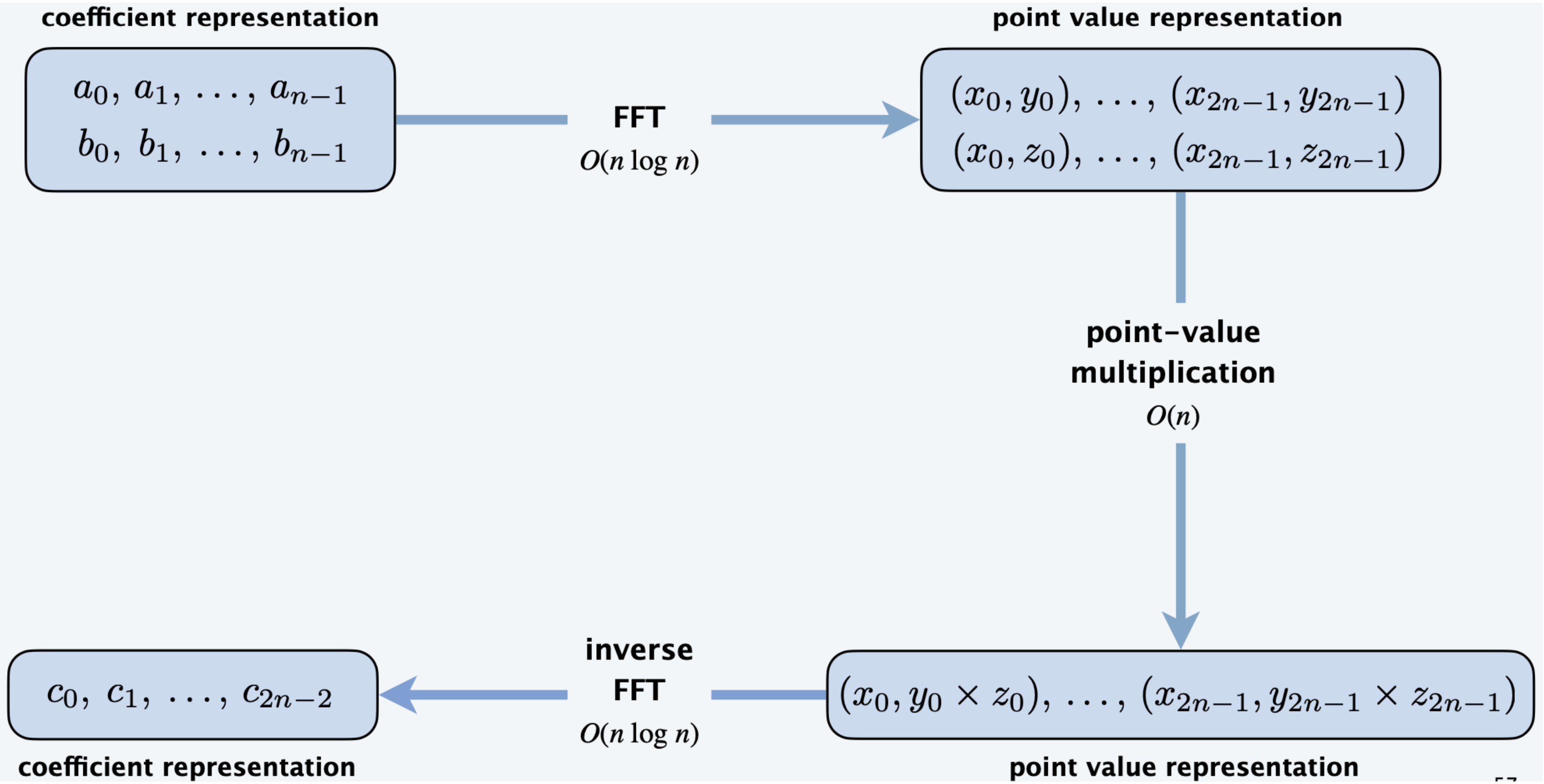
Representation	Multiply	Evaluate
coefficient	$O(n^2)$	$O(n)$
point-value	$O(n)$	$O(n^2)$

- **Goal.** Efficient conversion between two representations $\Rightarrow$ all ops fast.

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Converting between Two Representations

- **Application.** Polynomial multiplication (coefficient representation).



Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Converting between Two Representations

- **Coefficient ⇒ point-value.** Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.
  - ➢ Running time: $O(n^2)$ via matrix-vector multiplication or n Horner's
- **Point-value ⇒ coefficient.** Given n distinct points $x_0, \ldots, x_{n-1}$ and values $y_0, \ldots, y_{n-1}$, find unique polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$ that has given values at given points.

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

$O(n^3)$ via Gaussian elimination
$O(n^{2.37})$ via fast matrix multiplication

Vandermonde matrix is invertible iff x_i s are distinct

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Coefficient to Point-Value: Intuition

- **Coefficient ⇒ point-value.** Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$. ⟵ we get to choose these points!
- **Divide.** [Cooley-Tukey] Break up polynomial into even and odd degrees.
  - $A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7$
  - $A_{even}(x) = a_0 + a_2x + a_4x^2 + a_6x^3$ $\quad A_{odd}(x) = a_1 + a_3x + a_5x^2 + a_7x^3$
  - $A(x) = A_{even}(x^2) + x\, A_{odd}(x^2)$
  - $A(-x) = A_{even}(x^2) - x\, A_{odd}(x^2)$

can also divide into low and high degrees [Sande-Tukey]

- **Intuition.** Choose two points to be ± 1.
  - $A(1) = A_{even}(1) + 1\, A_{odd}(1)$
  - $A(-1) = A_{even}(1) - 1\, A_{odd}(1)$

can evaluate polynomial of degree $n - 1$ at 2 points by evaluating 2 polynomials of degree $\frac{1}{2} n - 1$ at 1 point

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Coefficient to Point-Value: Intuition

- **Coefficient ⇒ point-value.** Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$. ⟵ we get to choose these points!
- **Divide.** [Cooley-Tukey] Break up polynomial into even and odd degrees.
  - $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7$
  - $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3$ $\qquad A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3$
  - $A(x) = A_{even}(x^2) + x\, A_{odd}(x^2)$
  - $A(-x) = A_{even}(x^2) - x\, A_{odd}(x^2)$

can also divide into low and high degrees [Sande-Tukey]

- **Intuition.** Choose four complex points to be $\pm 1, \pm i$.
  - $A(1) = A_{even}(1) + 1\, A_{odd}(1)$
  - $A(-1) = A_{even}(1) - 1\, A_{odd}(1)$
  - $A(i) = A_{even}(-1) + i\, A_{odd}(-1)$
  - $A(-i) = A_{even}(-1) - i\, A_{odd}(-1)$

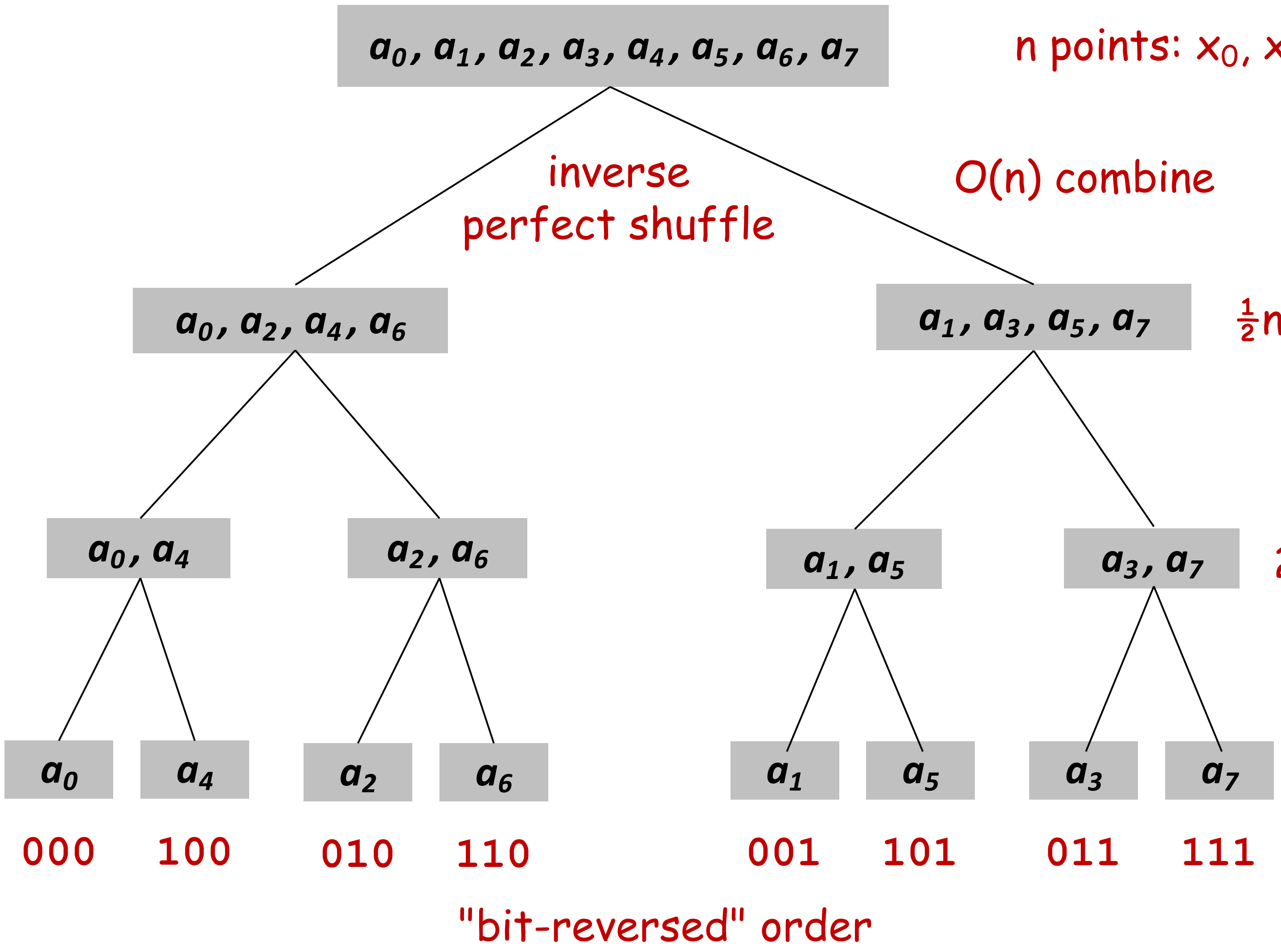
can evaluate polynomial of degree $n - 1$ at 4 points by evaluating 2 polynomials of degree ½ $n - 1$ at 2 points

can evaluate polynomial of degree $n - 1$ at n points by evaluating 2 polynomials of degree ½ $n - 1$ at $\boldsymbol{n/2}$ points

PEARSON Addison Wesley

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Coefficient to Point-Value: Recursion Tree



n points: $x_0, x_1, \ldots, x_{n-1}$ $\quad x_k^2 = x_{k+\frac{1}{2}n}^2$

$\frac{1}{2}$n points: $x_0^2, x_1^2, \ldots, x_{\frac{1}{2}n-1}^2$

2 points: $x_0^{\frac{1}{2}n}, x_1^{\frac{1}{2}n}$ $\quad (x_0^{\frac{1}{2}n})^2 = (x_1^{\frac{1}{2}n})^2$

1 point: $x_0^n = ?$ $\quad x_k^n = ?$

$x_0^n = \ldots = x_{n-1}^n = 1$

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Discrete Fourier Transform (DFT)

- **Coefficient ⇒ point-value.** Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$. ⟵ we get to choose these points!
- **Key idea:** choose $x_k = \omega^k$ where $\omega = e^{2\pi i / n}$ is principal n^{th} root of unity.

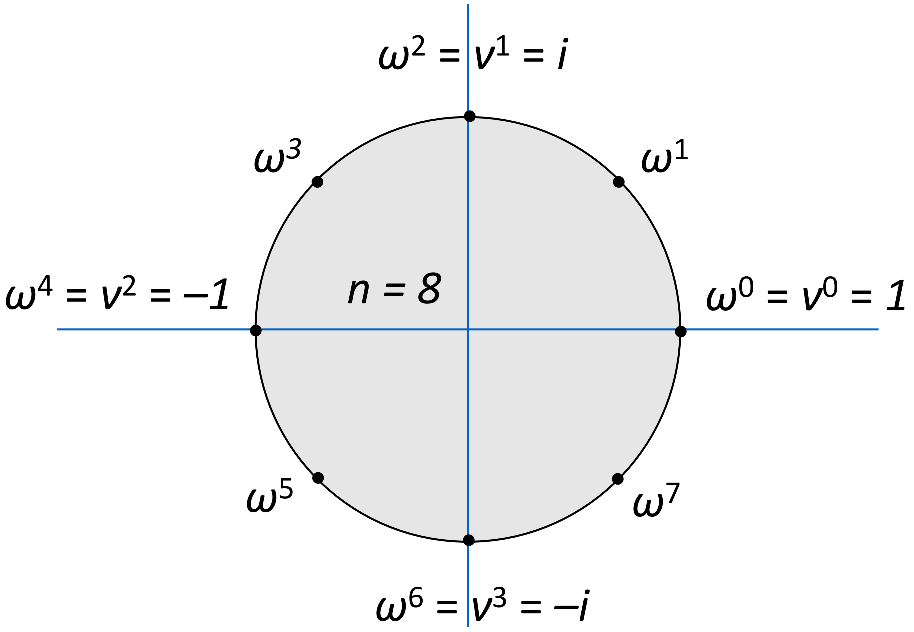
$y_k = A(\omega^k)$ ⟶

$$\underbrace{\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix}}_{\text{DFT}} = \underbrace{\begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}}_{\text{Fourier matrix } F_n} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

PEARSON Addison Wesley
Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Roots of Unity

- **Def.** An n^{th} root of unity is a complex number x such that $x^n = 1$.
- **Fact.** The n^{th} roots of unity are: $\omega^0, \omega^1, \ldots, \omega^{n-1}$ where $\omega = e^{2\pi i / n}$.
- **Pf.** $(\omega^k)^n = (e^{2\pi i k / n})^n = (e^{\pi i})^{2k} = (-1)^{2k} = 1$ common alternative: $\omega = e^{-2\pi i / n}$
- **Fact.** The $\frac{1}{2}n^{th}$ roots of unity are: $\nu^0, \nu^1, \ldots, \nu^{n/2-1}$ where $\nu = \omega^2 = e^{4\pi i / n}$.



Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Fast Fourier Transform (FFT)

- **Goal.** Evaluate a degree $n - 1$ polynomial $A(x) = a_0 + \ldots + a_{n-1} x^{n-1}$ at its n^{th} roots of unity: $\omega^0, \omega^1, \ldots, \omega^{n-1}$.
- **Divide.** Break up polynomial into even and odd powers.
  - $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + \ldots + a_{n-2} x^{n/2-1}$
  - $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + \ldots + a_{n-1} x^{n/2-1}$
  - $A(x) = A_{even}(x^2) + x A_{odd}(x^2)$
  - $A(-x) = A_{even}(x^2) - x A_{odd}(x^2)$
- **Conquer.** Evaluate $A_{even}(x)$, $A_{odd}(x)$ at $\frac{1}{2}n^{\text{th}}$ roots of unity: $v^0, v^1, \ldots, v^{n/2-1}$
- **Combine.** ($v = \omega^2$ and $\omega^{n/2} = -1$)
  - $A(\omega^k) = A_{even}(v^k) + \omega^k A_{odd}(v^k), \quad 0 \le k < n/2$
  - $A(\omega^{k+n/2}) = A_{even}(v^k) - \omega^k A_{odd}(v^k), \quad 0 \le k < n/2$

PEARSON Addison Wesley

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Fast Fourier Transform (FFT): Algorithm

- **FFT algorithm:**

```
FFT(n,a₀,a₁,…,aₙ₋₁) {
   if (n = 1) return a₀
   (e₀,e₁,…,eₙ/₂₋₁) = FFT(n/2,a₀,a₂,a₄,…,aₙ₋₂)
   (d₀,d₁,…,dₙ/₂₋₁) = FFT(n/2,a₁,a₃,a₅,…,aₙ₋₁)

   for k = 0 to n/2 - 1 {
      ωᵏ = e^(2πik/n)
      yₖ     = eₖ + ωᵏ dₖ
      yₖ₊ₙ/₂ = eₖ - ωᵏ dₖ
   }

   return (y₀,y₁,…,yₙ₋₁)
}
```

$2T(n / 2)$

$O(n)$

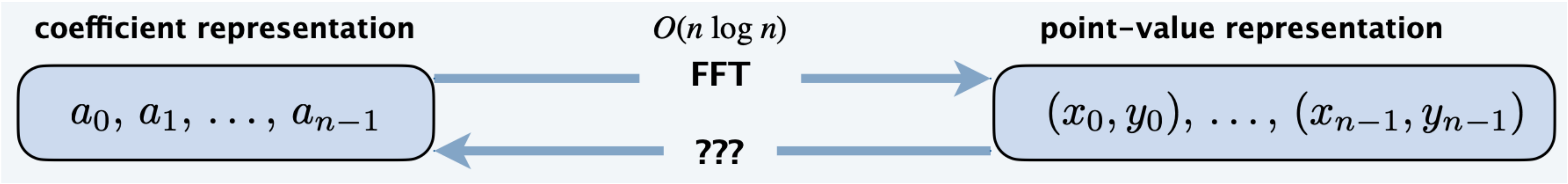
PEARSON Addison Wesley
Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

FFT Summary

- **Theorem.** The FFT algorithm evaluates a degree $n - 1$ polynomial at each of the n^{th} roots of unity in $O(n \log n)$ operations.

  assume n is a power of 2

- **Pf.** $T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$
- **Q.** What about space complexity?
- **A.** $O(n)$



Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Point-Value to Coefficient: Inverse DFT

- **Point-value ⇒ coefficient.** Given *n distinct* points $x_0, \ldots, x_{n-1}$ and values $y_0, \ldots, y_{n-1}$, find unique polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$ that has given values at given points.

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}^{-1} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

inverse DFT

Fourier matrix inverse F_n^{-1}

PEARSON Addison Wesley

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Inverse Discrete Fourier Transform

- **Claim.** Inverse of Fourier matrix F_n is given by following formula:

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

- **Consequence.** To compute the inverse FFT, apply the same algorithm but use $\omega^{-1} = e^{-2\pi i / n}$ as principal n^{th} root of unity (and divide the result by n).

PEARSON Addison Wesley
Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Inverse FFT: Proof of Correctness

- **Claim.** F_n and G_n are inverses.
- **Pf.**

$$(F_n G_n)_{kk'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{kj} \omega^{-jk'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{(k-k')j} = \begin{cases} 1 & \text{if } k = k' \\ 0 & \text{otherwise} \end{cases}$$

summation lemma (below)

- **Summation lemma.** Let ω be a principal n^{th} root of unity. Then

$$\sum_{j=0}^{n-1} \omega^{kj} = \begin{cases} n & \text{if } k \equiv 0 \pmod{n} \\ 0 & \text{otherwise} \end{cases}$$

- **Pf.**
  - If k is a multiple of n then $\omega^k = 1$, so the series sums to n.
  - Each n^{th} root of unity ω^k is a root of $x^n - 1 = (x - 1)(1 + x + x^2 + \ldots + x^{n-1})$.
  - If $\omega^k \neq 1$, then $1 + \omega^k + \omega^{k(2)} + \ldots + \omega^{k(n-1)} = 0$, so the series sums to 0. ∎

PEARSON Addison Wesley
Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Inverse FFT: Algorithm

- **Inverse FFT algorithm:**

```
Inverse-FFT(n,y0,y1,…,yn-1) {
   if (n == 1) return a0
   (e0,e1,…,en/2-1) = Inverse-FFT(n/2,y0,y2,y4,…,yn-2)
   (d0,d1,…,dn/2-1) = Inverse-FFT(n/2,y1,y3,y5,…,yn-1)

   for k = 0 to n/2 - 1 {
      ωk = e-2πik/n
      ak     = ek + ωk dk
      ak+n/2 = ek - ωk dk
   }
   return (a0,a1,…,an-1)
}
Output: Inverse-FFT(n,y0,y1,…,yn-1) / n
```

$2T(n / 2)$

$O(n)$

PEARSON Addison Wesley

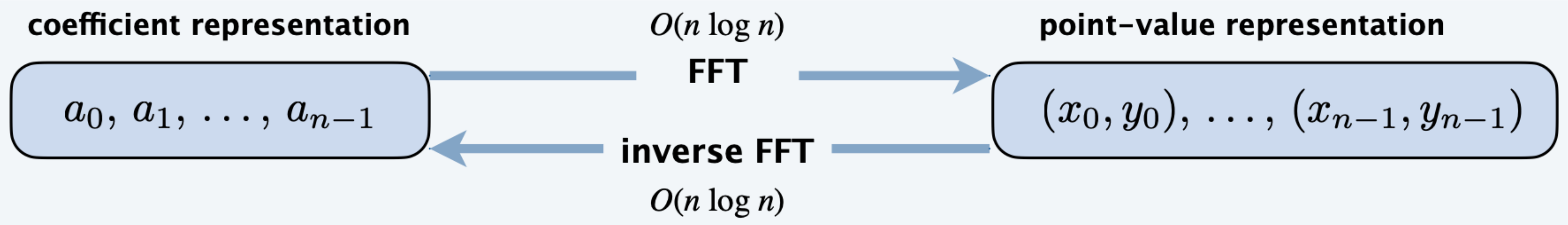
Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Inverse FFT Summary

- **Theorem.** The inverse FFT algorithm interpolates a degree *n – 1* polynomial at each of the n^{th} roots of unity in $O(n \log n)$ operations.

assume n is a power of 2

- **FFT + Inverse FFT.** Can convert between coefficient and point-value representations in $O(n \log n)$ arithmetic operations.

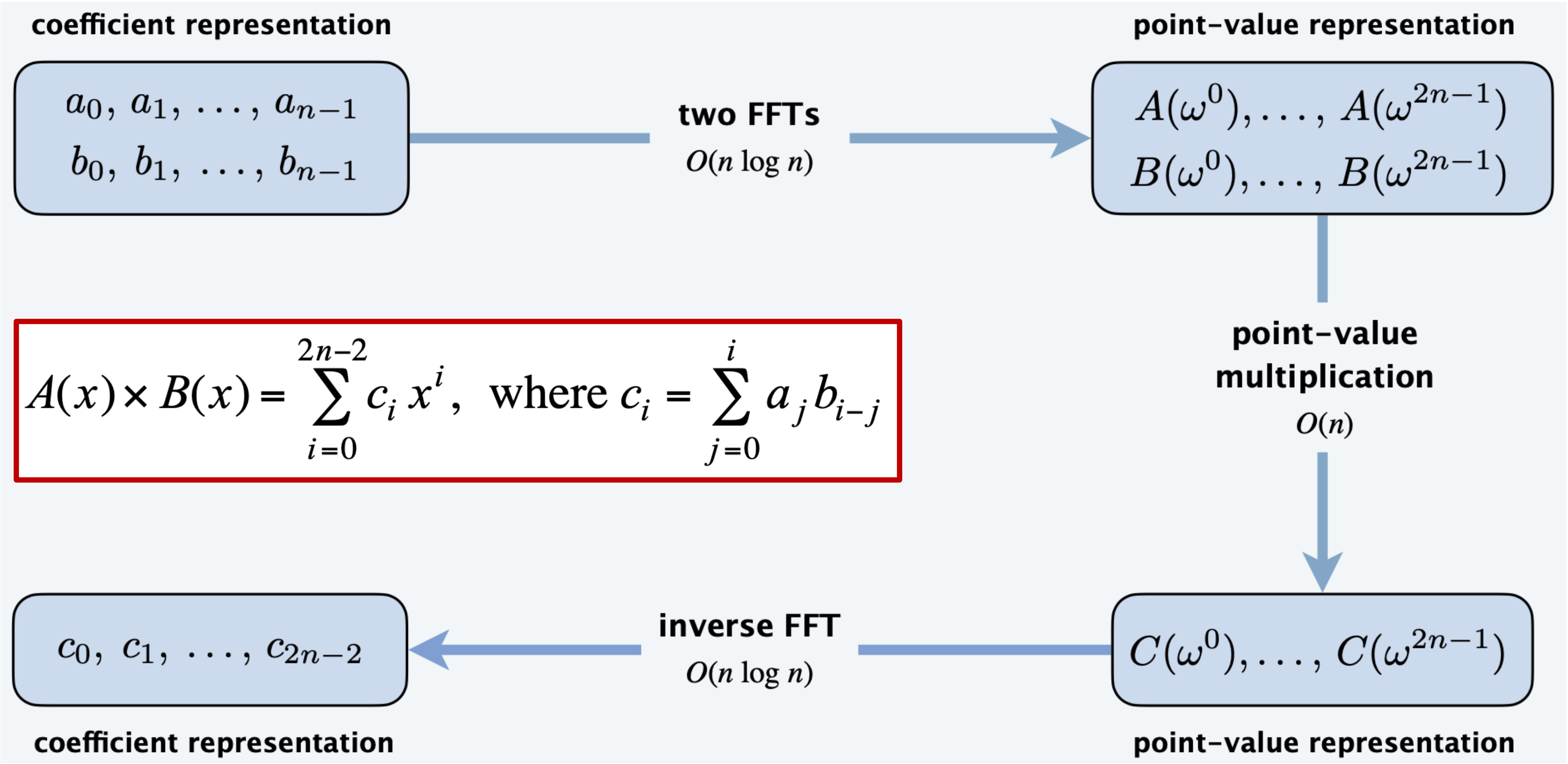


PEARSON Addison Wesley
Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Fast Polynomial Multiplication

- **Theorem.** Can multiply two degree $n - 1$ polynomials in $O(n \log n)$ steps.
- **Pf.**

pad 0 items to make n a power of 2



Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Integer Multiplication Revisited

- **Integer multiplication.** Given two n bit integers $a = a_{n-1} \ldots a_1 a_0$ and $b = b_{n-1} \ldots b_1 b_0$, compute their product $c = ab$.
- **Convolution algorithm:**
  - Form two polynomials $A(x)$, $B(x)$.
  - Note that $a = A(2)$, $b = B(2)$.
  - Compute $C(x) = A(x)\,B(x)$ and evaluate $C(2) = ab$.
  - Running time: $O(n \log n)$ floating-point operations ($O(n (\log n)^3)$ bit operations).

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

$$B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1}$$

- **Theory.** [Schönhage-Strassen 1971]
  - $O(n \log^2 n)$ bit operations over complex numbers (with $O(\log n)$ bit precision)
  - $O(n \log n \log \log n)$ bit operations over ring of integers (mod a Fermat number)
- **Practice.** GNU Multiple Precision Arithmetic Library (GMP) switches to FFT-based algorithms when n is large (≥ 5~$10K$)

PEARSON Addison Wesley
Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

FFT in Practice

- **FFT in the West (FFTW)** [Frigo-Johnson]
  - Optimized C library.
  - Features: DFT, DCT, real, complex, any size, any dimension.
  - Won 1999 Wilkinson Prize for Numerical Software.
  - Portable, competitive with vendor-tuned code.
- **Implementation details.**
  - Core algorithm is an in-place, nonrecursive version of Cooley–Tukey.
  - Instead of executing a fixed algorithm, it evaluates the hardware and uses a special-purpose compiler to generate an optimized algorithm catered to the "shape" of the problem.
  - Runs in $O(n \log n)$ time, even when n is prime.
  - Multidimensional FFTs.
  - Parallelism.

Reference: http://www.fftw.org

Slides by Kevin Wayne. Copyright © 2005 Pearson-Addison Wesley. All rights reserved.

Announcement

- **Assignment 3 has been released and the deadline is 2pm, April 17.**