Divide and Conquer

Closest Pair of Points

Closest Pair of Points

Closest pair. Given n points in the plane, find a pair with smallest Euclidean distance between them.

Fundamental geometric primitive.

- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, Euclidean MST, Voronoi.

fast closest pair inspired fast algorithms for these problems

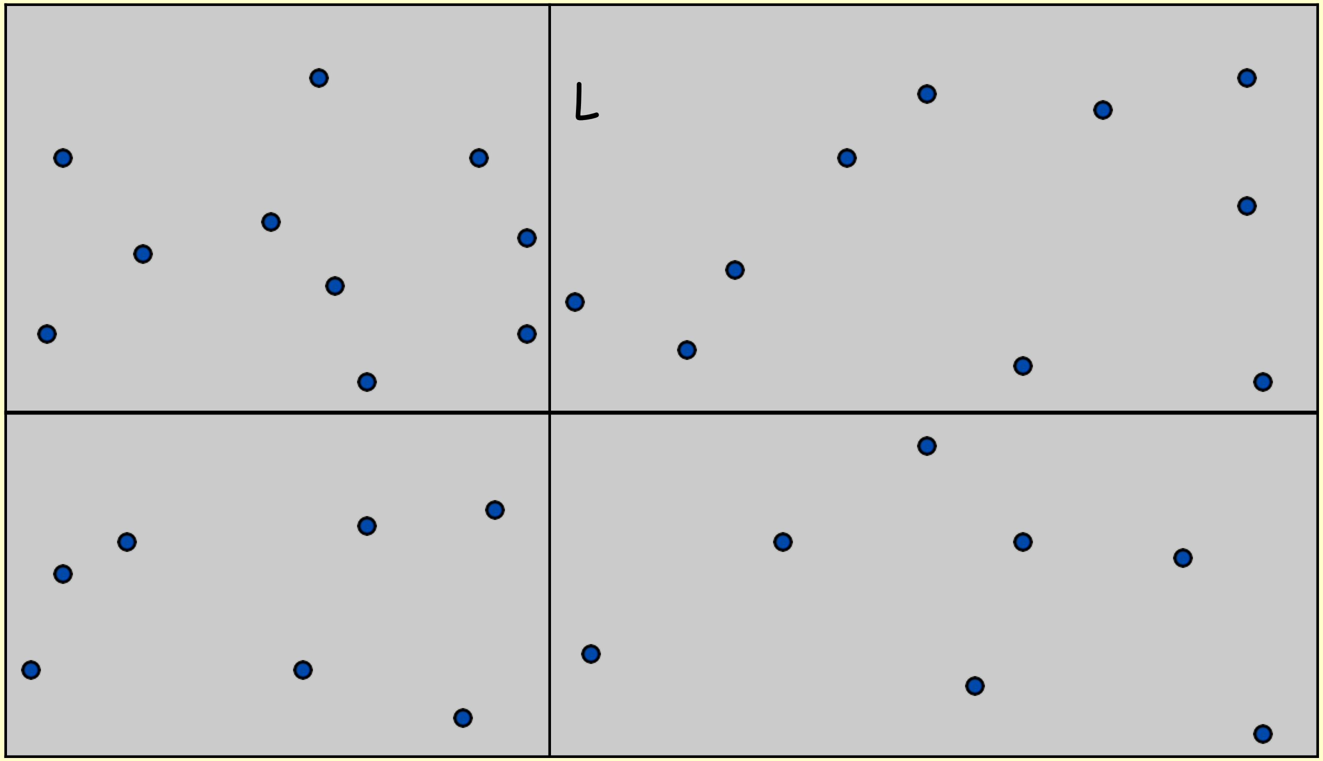
Brute force. Check all pairs of points p and q with $\Theta(n^2)$ comparisons. 1-D version. $O(n \log n)$ easy if points are on a line.

Assumption. No two points have same x coordinate.

to make presentation cleaner

Closest Pair of Points: First Attempt

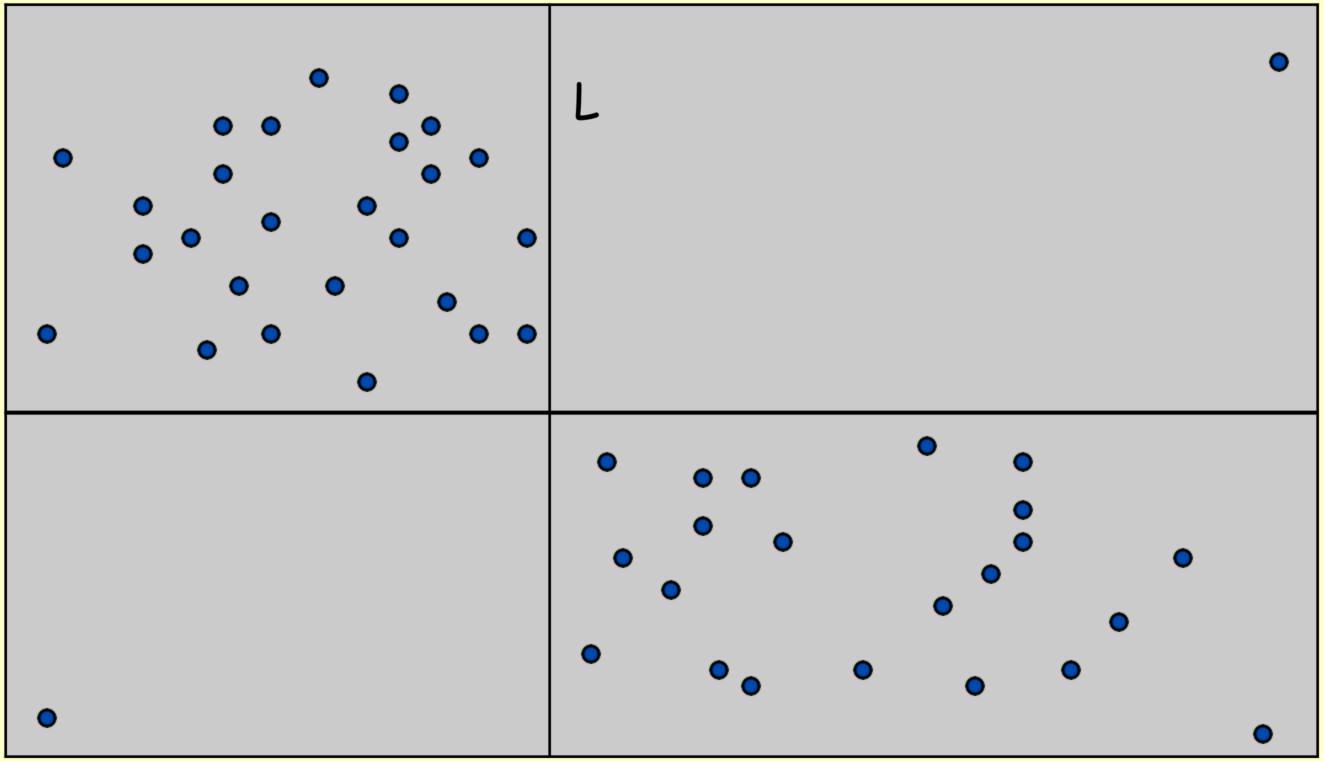
Divide. Sub-divide region into 4 quadrants.



Closest Pair of Points: First Attempt

Divide. Sub-divide region into 4 quadrants.

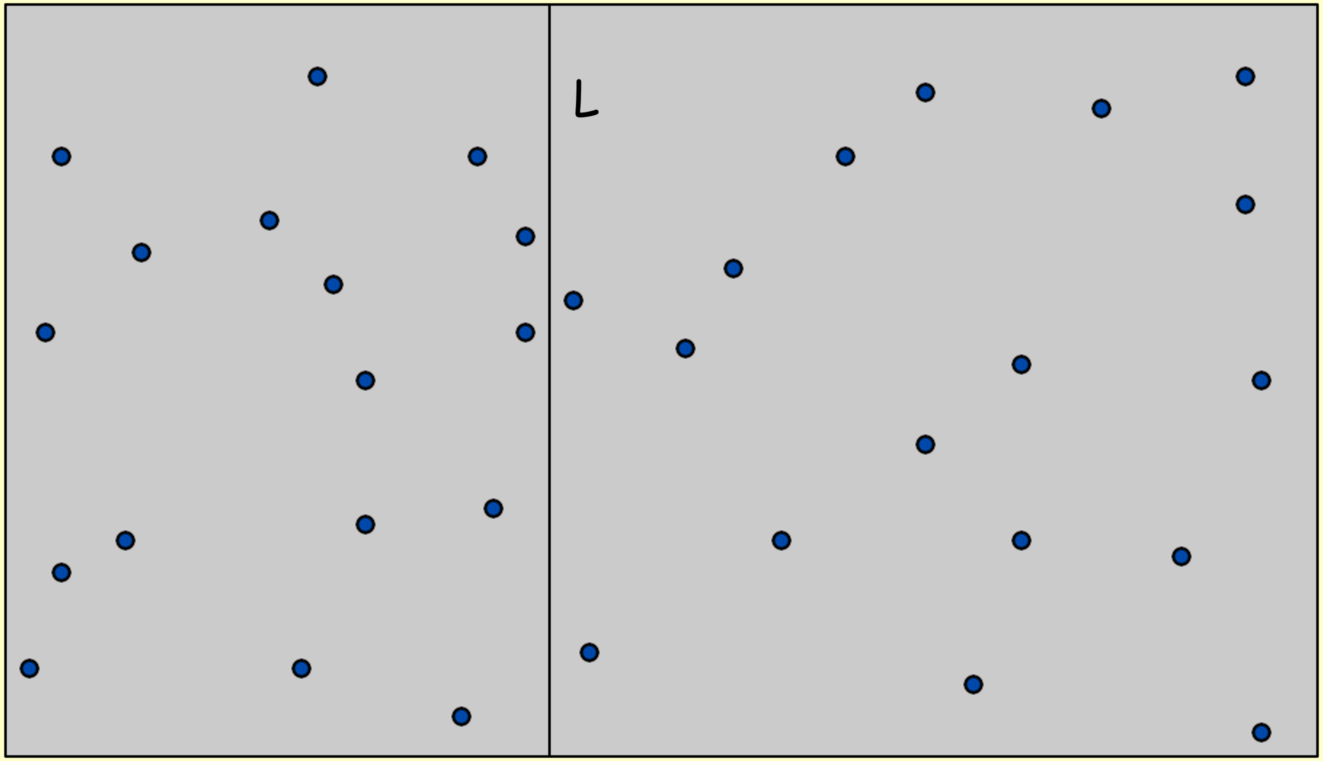
Obstacle. Impossible to ensure n/4 points in each piece.



Closest Pair of Points

Algorithm.

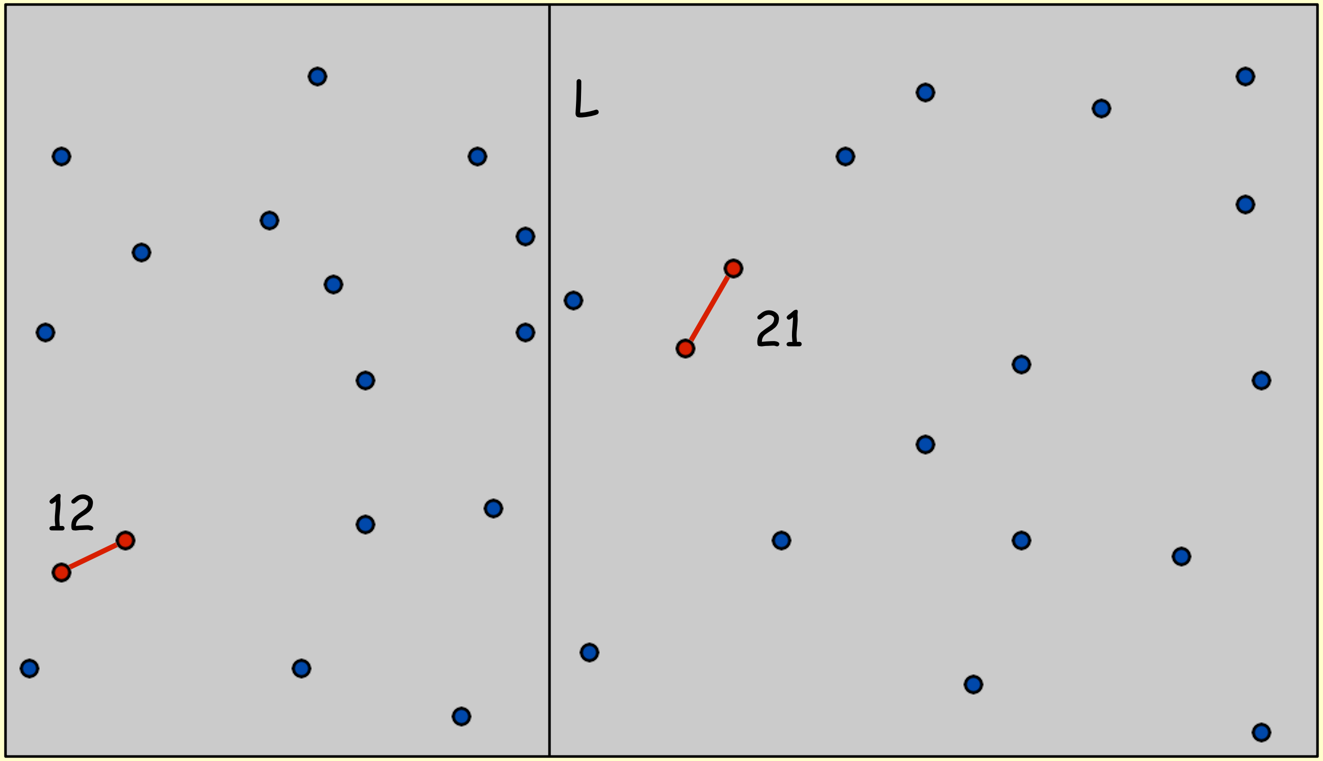
- Divide: draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.



Closest Pair of Points

Algorithm.

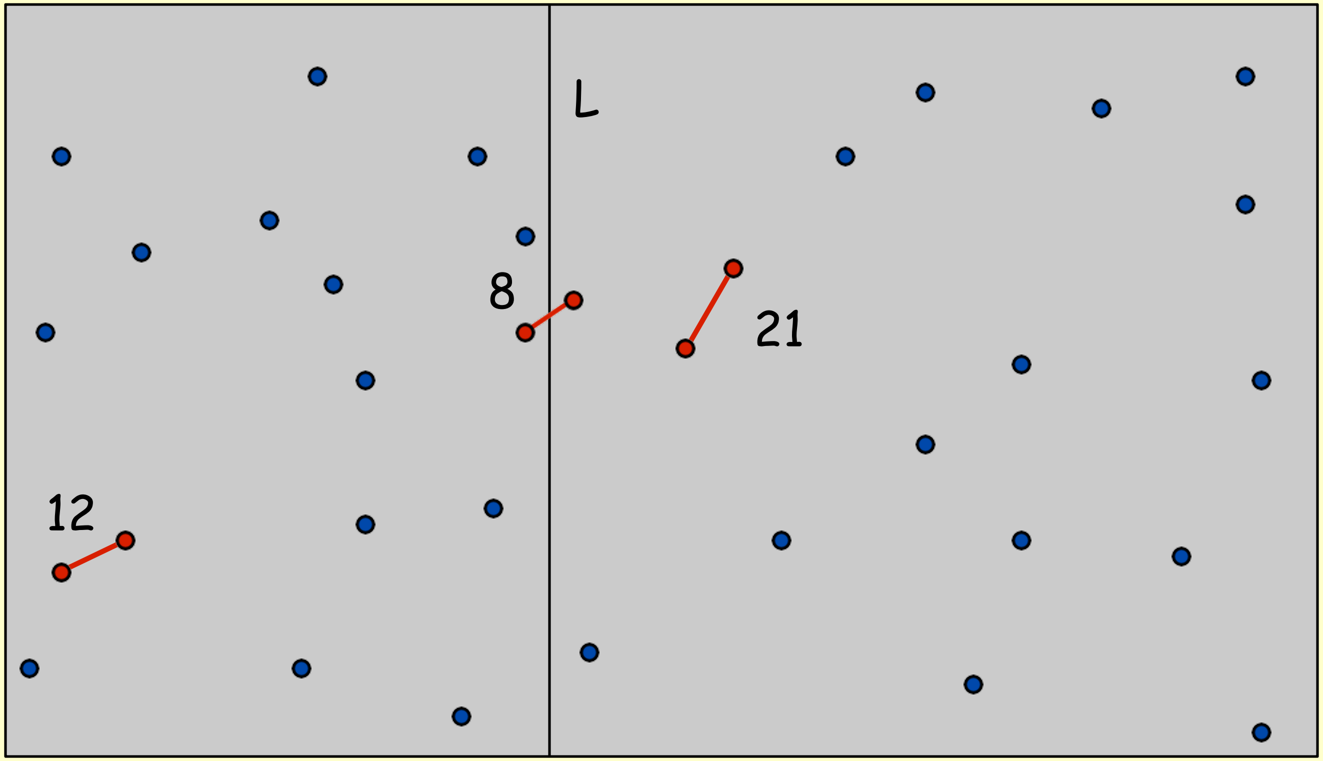
- Divide: draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.
- Conquer: find closest pair in each side recursively.



Closest Pair of Points

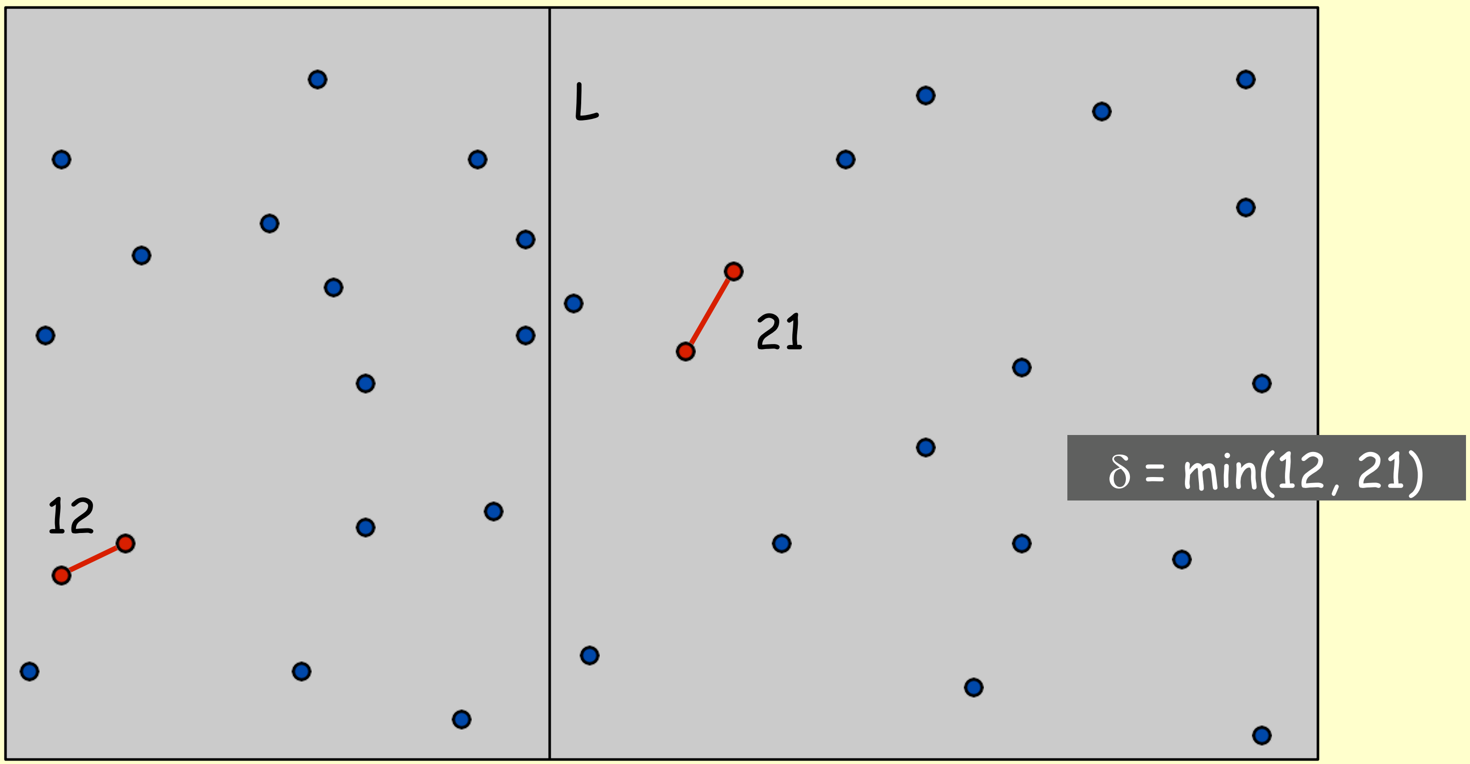
Algorithm.

- Divide: draw vertical line L so that roughly $\frac{1}{2}$n points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side. ← seems like $\Theta(n^2)$
- Return best of 3 solutions.



Closest Pair of Points

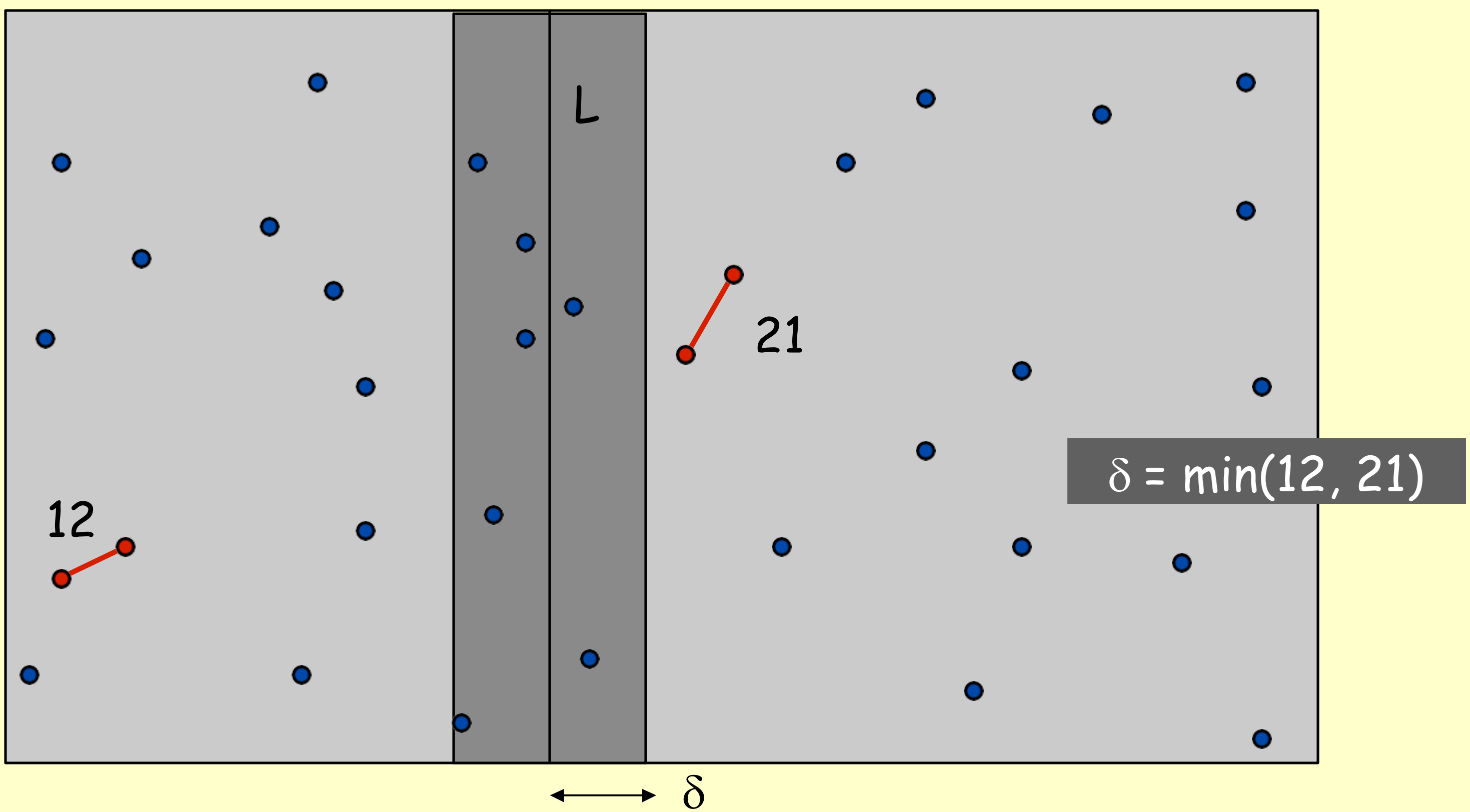
Find closest pair with one point in each side, assuming that distance < δ.



Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < δ.

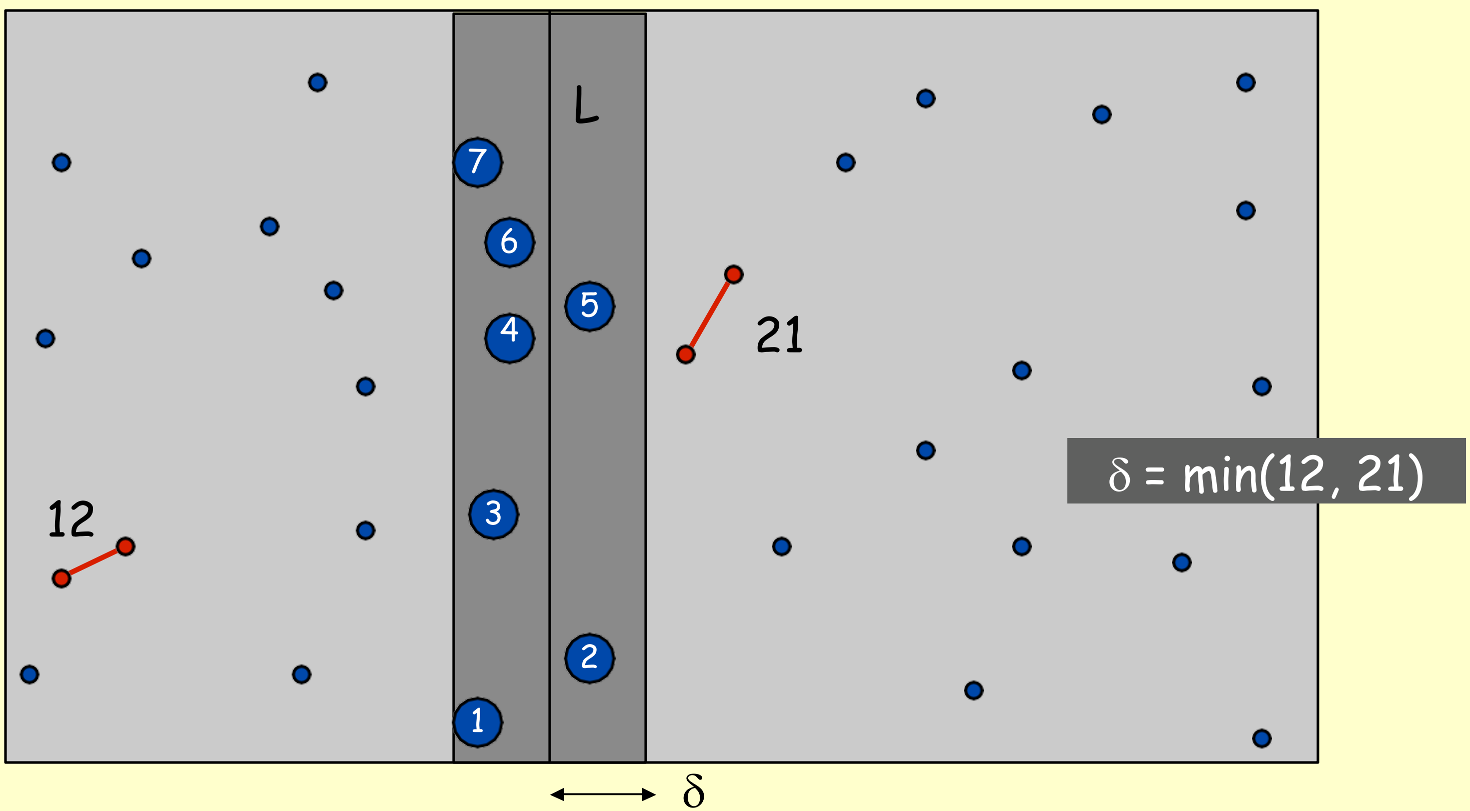
- Observation: only need to consider points within δ of line L.



Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < δ.

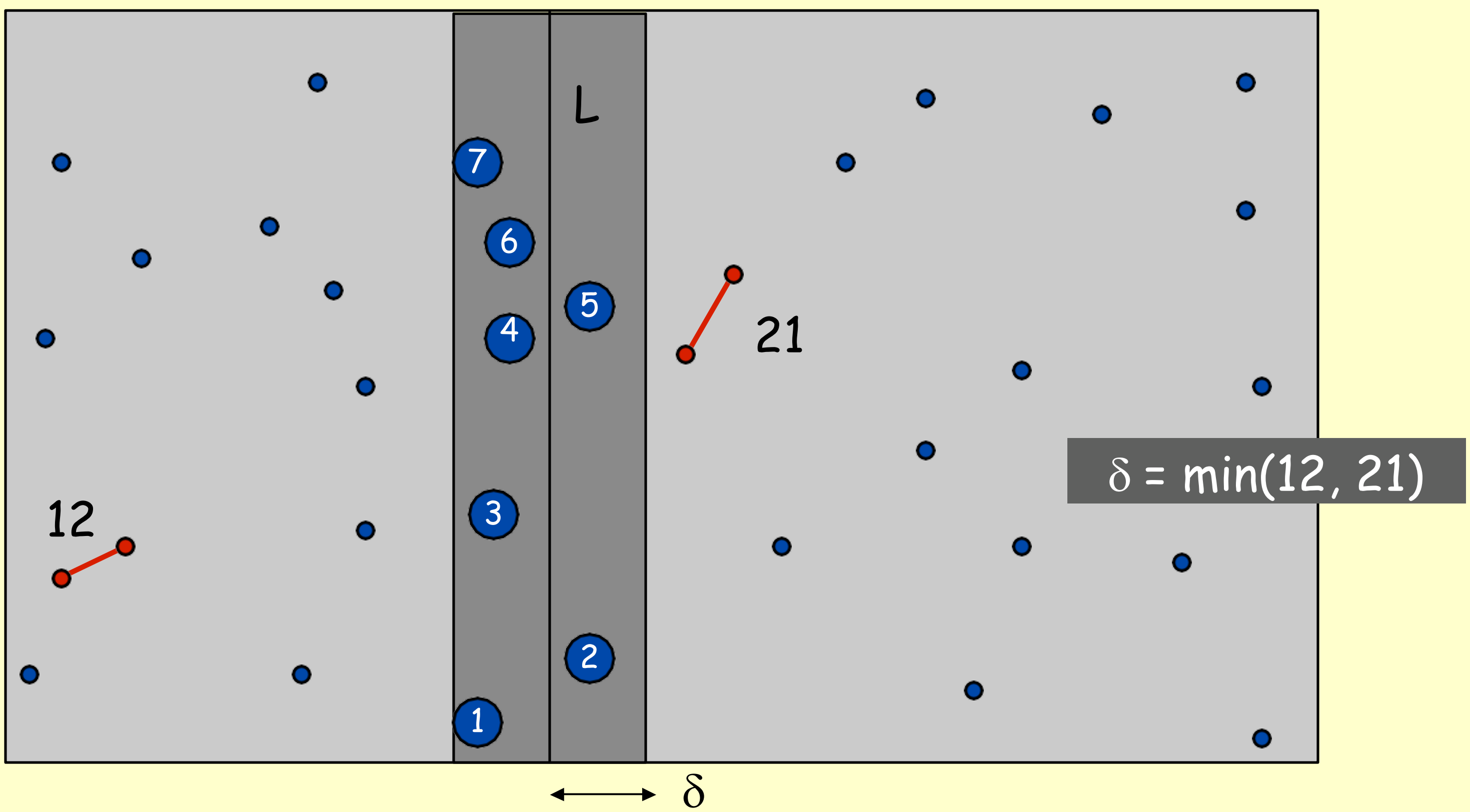
- Observation: only need to consider points within δ of line L.
- Sort points in 2δ-strip by their y coordinate.



Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < δ.

- Observation: only need to consider points within δ of line L.
- Sort points in 2δ-strip by their y coordinate.
- Only check distances of those within 11 positions in sorted list!



Closest Pair of Points

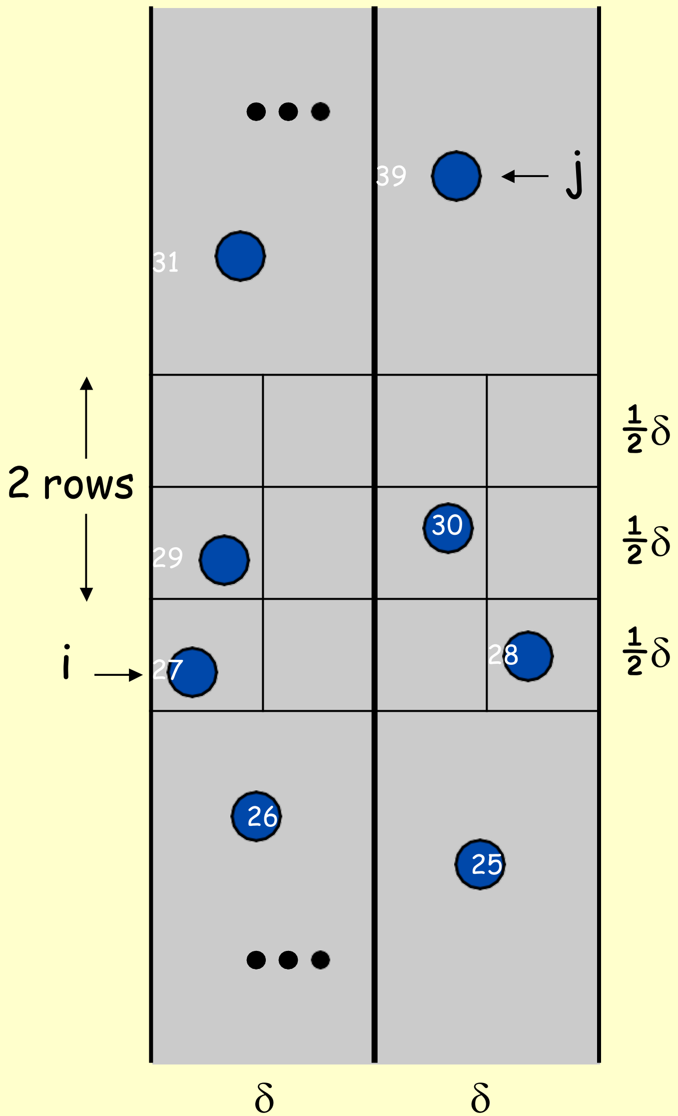
Def. Let s_i be the point in the 2δ-strip, with the i^{th} smallest y-coordinate.

Claim. If $|i - j| \geq 12$, then the distance between s_i and s_j is at least δ.

Pf.

- No two points lie in same $\frac{1}{2}\delta$-by-$\frac{1}{2}\delta$ box.
- Two points at least 2 rows apart have distance $\geq 2(\frac{1}{2}\delta)$. ▪

Fact. Still true if we replace 12 with 7.



Closest Pair Algorithm

```
Closest-Pair(p₁, …, pₙ) {
Compute separation line L such that half the points
are on one side and half on the other side.

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

   Delete all points further than δ from separation line L

   Sort remaining points by y-coordinate.

   Scan points in y-order and compare distance between
   each point and next 11 neighbors. If any of these
   distances is less than δ, update δ.

   return δ.
}
```

Running times, top to bottom: $O(n \log n)$ for computing the separation line, $2T(n / 2)$ for the two recursive calls, $O(n)$ for the delete step, $O(n \log n)$ for the sort and $O(n)$ for the scan.

Closest Pair of Points

Running time:

$$T(n) \le 2T\left(n/2\right) + O(n \log n) \Rightarrow T(n) = O(n \log^2 n)$$

Q. Can we achieve O(n log n)?

A. Yes. Don't sort points in strip from scratch each time.

- Each recursive returns two lists: all points sorted by y coordinate, and all points sorted by x coordinate.
- Sort by merging two pre-sorted lists.

$$T(n) \le 2T\left(n/2\right) + O(n) \Rightarrow T(n) = O(n \log n)$$

```
Closest-Pair(P)
  Construct Px and Py  (O(n log n) time)
  (p0*, p1*) = Closest-Pair-Rec(Px,Py)

Closest-Pair-Rec(Px, Py)
  If |P| ≤ 3 then
    find closest pair by measuring all pairwise distances
  Endif

  Construct Qx, Qy, Rx, Ry (O(n) time)
  (q0*,q1*) = Closest-Pair-Rec(Qx, Qy)
  (r0*,r1*) = Closest-Pair-Rec(Rx, Ry)

  δ = min(d(q0*,q1*), d(r0*,r1*))
  x* = maximum x-coordinate of a point in set Q
  L = {(x,y) : x = x*}
  S = points in P within distance δ of L.

  Construct Sy (O(n) time)
  For each point s ∈ Sy, compute distance from s
     to each of next 15 points in Sy
     Let s, s' be pair achieving minimum of these distances
     (O(n) time)

  If d(s,s') < δ then
     Return (s,s')
  Else if d(q0*,q1*) < d(r0*,r1*) then
     Return (q0*,q1*)
  Else
     Return (r0*,r1*)
  Endif
```

Divide and Conquer: Convolution and FFT

Convolution

- Convolution, is a mathematical operation on two functions f and g to produce a third function (f*g) that expresses how the shape of one is modified by the other.
- It is defined as the integral of the product of the two functions after one is reflected about the y-axis and shifted.

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau.$$

- The integral is evaluated for all values of shift, producing the convolution function.

Convolution

Applications:

- Image Processing
- Digital data processing
- Probability theory
- Neural Network
- Electrical Engineering
- Acoustics etc.

Convolution

- The convolution of two vectors, $u = (a_0, a_1, a_2 \ldots, a_{n-1})$ and $v = (b_0, b_1, b_2 \ldots, b_{m-1})$, represents the area of overlap under the points (curve) as v slides across u.
- $a * b$ is a vector with $m + n - 1$ coordinates, where coordinate k is equal to $$\sum_{\substack{(i,j):i+j=k \\ i<m,\, j<n}} a_i b_j.$$
- Or, if $u = (a_1, a_2 \ldots, a_n)$ and $v = (b_1, b_2 \ldots, b_m)$, and w = a*b, $$w(k) = \sum_j u(j)v(k - j + 1).$$
- The sum is over all the values of j that lead to legal subscripts for u(j) and v(k-j+1).
- Specifically j = max(1, k+1-n):1:min(k, m).

Convolution

- Example: u=(1, 1, 1); v = (1, 1, 0, 0, 0, 1, 1)
  u = (-1, 2, 3, -2, 0, 1, 2,); v = (2, 4, -1, 1)
- Algebraically, convolution is the equivalent operation as multiplying polynomials whose coefficients are the elements of *u* and *v*.
- The direct approach to compute the convolution involves calculating the product $a_i b_j$ for every pair (i, j) and takes O(n^2) arithmetic operations if m = n.
- Can we reduce it?
  - Yes, using Fast Fourier Transform (FFT).

Fast Fourier Transform

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

Alternate viewpoint. Fast way to multiply and evaluate polynomials.

- A polynomial A(x) can be written in the following forms:

$$\begin{aligned} A(x) &= a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} \\ &= \sum_{k=0}^{n1} a_k x^k \\ &= \langle a_0, a_1, a_2, \ldots, a_{n-1} \rangle \quad \text{(coefficient vector)} \end{aligned}$$

The degree of A is n–1.

Operations on polynomials

- There are three primary operations for polynomials.

1. **Evaluation**: Given a polynomial A(x) and a number x_0, compute A(x_0).

2. Addition: Given two polynomials A(x)and B(x), compute C(x) = A(x)+ B(x) ∀x. This takes O(n) time using basic arithmetic, because $c_k = a_k + b_k$.

3. Multiplication: Given two polynomials A(x) and B(x), compute C(x) = A(x)·B(x) ∀x.

Then $c_k = \sum_{j=0}^{k} a_j b_{k-j}$ for $0 \leq k \leq 2(n-1)$

- This multiplication is equivalent to a convolution of the vectors A and reverse(B). The convolution is the inner product of all relative shifts, an operation also useful for smoothing etc. in digital signal processing.

Polynomials: Coefficient Representation

Polynomial.[coefficient representation]

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

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

Add. $O(n)$ arithmetic operations.

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

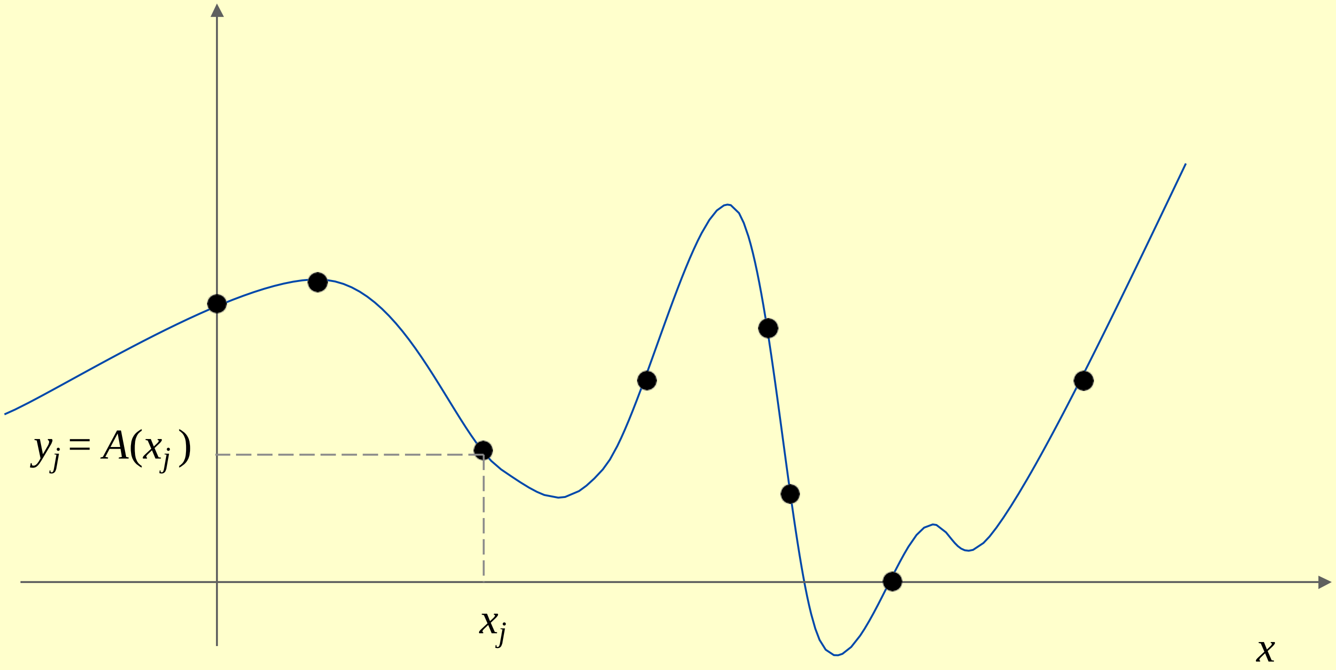
Evaluate. $O(n)$ using Horner's method.

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

Multiply (convolve). $O(n^2)$ using brute force.

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

Polynomials: Point-Value Representation



Polynomials: Point-Value Representation

Polynomial. [point-value representation]

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

Add. $O(n)$ arithmetic operations.

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

Multiply (convolve). $O(n)$, but need $2n$-1 points.

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

Evaluate. $O(n^2)$ using Lagrange's formula.

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

Converting Between Two Polynomial Representations

Tradeoff. 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.

$$a_0, a_1, \ldots, a_{n-1} \rightleftarrows (x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$$

coefficient representation

point-value representation

Converting Between Two Representations: Brute Force

Coefficient $\Rightarrow$ point-value. Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.

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

Running time. $O(n^2)$ for matrix-vector multiply (or n Horner's).

Converting Between Two Representations: Brute Force

Point-value $\Rightarrow$ 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.

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

Vandermonde matrix is invertible iff x_i distinct

Running time. $O(n^3)$ for Gaussian elimination.

or $O(n^{2.376})$ via fast matrix multiplication

Divide-and-Conquer

Decimation in time. Break polynomial up into even and odd terms.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$
- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3.$
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3.$
- $A(x) = A_{even}(x^2) + x\, A_{odd}(x^2).$

Coefficient to Point-Value Representation: Intuition

Coefficient $\Rightarrow$ point-value. Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.

Divide. Break polynomial up into even and odd coefficients.

$$A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$$

- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3.$
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3.$
- $A(x) = A_{even}(x^2) + x\, A_{odd}(x^2).$
- $A(-x) = A_{even}(x^2) - x\, A_{odd}(x^2).$

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 $\leq n$ at 2 points by evaluating two polynomials of degree $\leq \frac{1}{2} n$ at 1 point.

Coefficient to Point-Value Representation: Intuition

Coefficient $\Rightarrow$ point-value. Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.

Divide. Break polynomial up into even and odd coefficients.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7$.
- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3$.
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3$.
- $A(x) = A_{even}(x^2) + x\, A_{odd}(x^2)$.
- $A(-x) = A_{even}(x^2) - x\, A_{odd}(x^2)$.

Intuition. Choose four complex points to be ± 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 $\leq n$ at 4 points by evaluating two polynomials of degree $\leq \frac{1}{2}n$ at 2 points.

Discrete Fourier Transform

Coefficient $\Rightarrow$ point-value. Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.

Key idea. Choose $x_k = \omega^k$ where ω is principal n^{th} root of unity. Where $k = 0, 1, \ldots, n-1$

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

DFT

Fourier matrix F_n

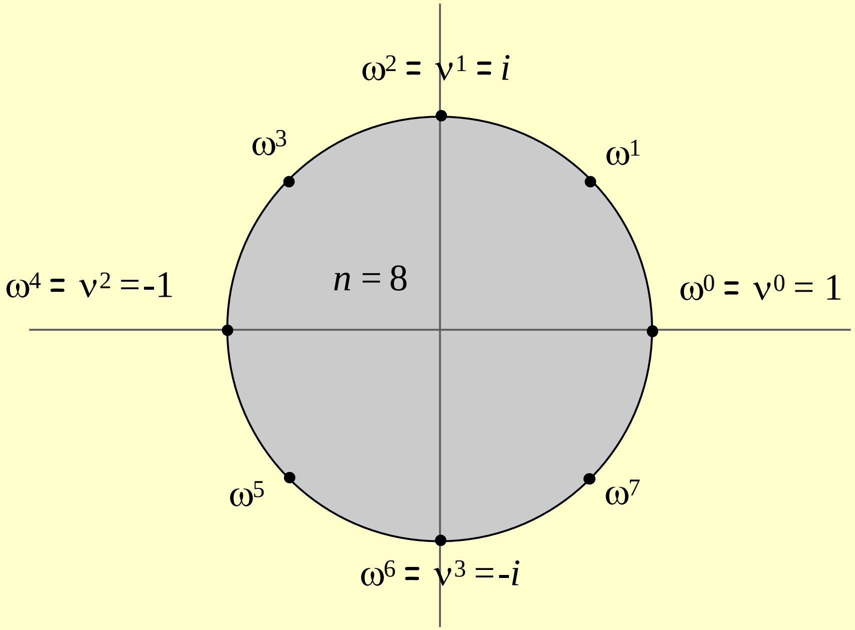
Roots of Unity

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. $(\omega^k)^n = (e^{2\pi i k / n})^n = (e^{\pi i})^{2k} = (-1)^{2k} = 1$.

Fact. The $\frac{1}{2}n^{th}$ roots of unity are: $\nu^0, \nu^1, \ldots, \nu^{n/2-1}$ where $\nu = \omega^2 = e^{4\pi i / n}$.



Fast Fourier Transform

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 terms.

- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + \ldots + a_{n-2} x^{n/2 - 1}$.
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + \ldots + a_{n-1} x^{n/2 - 1}$.
- $A(x) = A_{even}(x^2) + x A_{odd}(x^2)$.

Conquer. Evaluate $A_{even}(x)$ and $A_{odd}(x)$ at the $\frac{1}{2}n^{th}$ roots of unity: $\nu^0, \nu^1, \ldots, \nu^{n/2-1}$.

Combine.

$\nu^k = (\omega^k)^2$

- $A(\omega^k) = A_{even}(\nu^k) + \omega^k A_{odd}(\nu^k), \quad 0 \le k < n/2$
- $A(\omega^{k+ \frac{1}{2}n}) = A_{even}(\nu^k) - \omega^k A_{odd}(\nu^k), \quad 0 \le k < n/2$

$\nu^k = (\omega^{k + \frac{1}{2}n})^2$ $\quad\omega^{k+ \frac{1}{2}n} = -\omega^k$

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 ←  e2πik/n
      yk     ← ek +   ωk dk
      yk+n/2 ← ek -   ωk dk
   }

   return (y0,y1,…,yn-1)
}
```

FFT Summary

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

assumes n is a power of 2

Running time.

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

$$a_0, a_1, \ldots, a_{n-1} \xrightarrow{O(n \log n)} (\omega^0, y_0), \ldots, (\omega^{n-1}, y_{n-1})$$

coefficient representation

point-value representation

Inverse Discrete Fourier Transform

Point-value $\Rightarrow$ 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 $(F_n)^{-1}$

Inverse DFT

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

Inverse FFT: Algorithm

```
ifft(n, y_0,y_1,…,y_{n-1}) {
   if (n == 1) return y_0

   (e_0,e_1,…,e_{n/2-1}) ← FFT(n/2,  y_0,y_2,y_4,…,y_{n-2})
   (d_0,d_1,…,d_{n/2-1}) ← FFT(n/2,  y_1,y_3,y_5,…,y_{n-1})

   for k= 0 to n/2 - 1 {
      ω^k ← e^{-2πik/n}
      y_{k+n/2}  ← (e_k + ω^k d_k) / n
      y_{k+n/2}  ← (e_k - ω^k d_k) / n
   }

   return (a_0,a_1,…,a_{n-1})
}
```

Inverse FFT Summary

Theorem. Inverse FFT algorithm interpolates a degree n-1 polynomial given values at each of the n^{th} roots of unity in $O(n \log n)$ steps.

assumes n is a power of 2

$$\begin{tikzcd}[column sep=huge] a_0, a_1, \ldots, a_{n-1} \arrow[r, shift left, "O(n \log n)"] & (\omega^0, y_0), \ldots, (\omega^{n-1}, y_{n-1}) \arrow[l, shift left, "O(n \log n)"] \end{tikzcd}$$

coefficient representation

point-value representation

Polynomial Multiplication

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

pad with 0s to make n a power of 2

coefficient representation

$a_0, a_1, \ldots, a_{n\text{-}1}$

$b_0, b_1, \ldots, b_{n\text{-}1}$

2 FFTs $O(n \log n)$

$A(\omega^0), \ldots, A(\omega^{2n-1})$

$B(\omega^0), \ldots, B(\omega^{2n-1})$

point-value multiplication

$O(n)$

$C(\omega^0), \ldots, C(\omega^{2n-1})$

inverse FFT $O(n \log n)$

coefficient representation

$c_0, c_1, \ldots, c_{2n\text{-}2}$