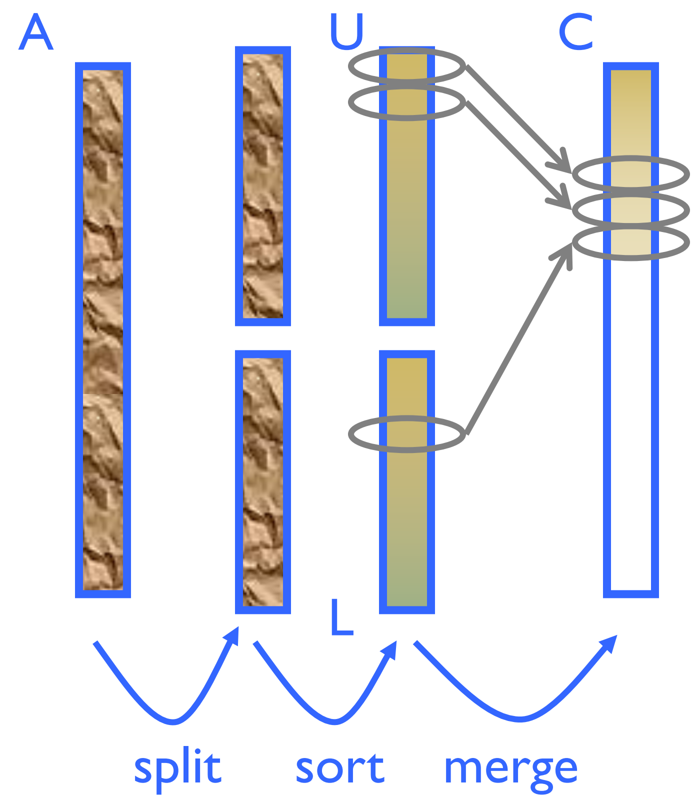
Divide & Conquer

- Reduce problem to one or more sub-problems of the same type
- Typically, each sub-problem is at most a constant fraction of the size of the original problem
- Subproblems typically disjoint
- Combine solutions
- Examples:
  - Mergesort, Binary Search, Strassen's Algorithm, Quicksort (roughly)

```
MS(A: array[1..n]) returns array[1..n] {
    If(n=1) return A;
    New U:array[1:n/2] = MS(A[1..n/2]);
    New L:array[1:n/2] = MS(A[n/2+1..n]);
    Return(Merge(U,L));
    }

Merge(U,L: array[1..n]) {
    New C: array[1..2n];
    a=1; b=1;
    For i = 1 to 2n
        C[i] = "smaller of U[a], L[b] and correspondingly a++ or b++";
    Return C;
    }
```



Alternative "divide & conquer" algorithm:

Sort n-1

Sort last 1

Merge them

$T(n) = T(n-1)+T(1)+n$ for $n > 2$

$T(1) = 0$

Solution: $n + (n-1) + (n-2) \ldots = \Theta(n^2)$

Suppose we've already invented Bubble-Sort, taking time n^2

Try *Just One Level* of divide & conquer:

Bubble-Sort(first n/2 elements)

Bubble-Sort(last n/2 elements)

Merge results

Time: $2\,(n/2)^2 + n = n^2/2 + n \ll n^2$

D&C in a nutshell

Almost twice as fast!

“two halves are better than a whole”

Two problems of half size are *better* than one full-size problem, even given O(n) overhead of recombining, since the base algorithm has *super-linear* complexity.

“the more dividing and conquering, the better”

Two levels of D&C would be almost 4 times faster, 3 levels almost 8, etc., even though overhead is growing.
Best is usually full recursion down to some small constant size (balancing "work" vs "overhead").

In the limit: you’ve just rediscovered mergesort!

unbalanced division less good, but still good

Bubble-sort improved with 0.1/0.9 split:

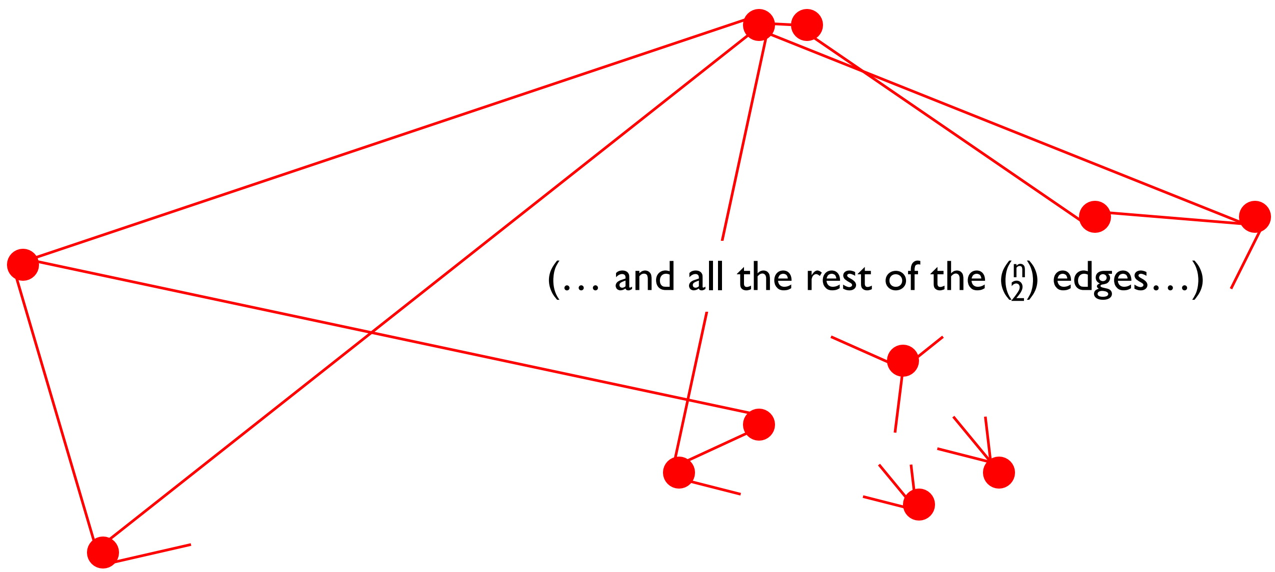
$(.1n)^2 + (.9n)^2 + n = .82n^2 + n$

- The 18% savings compounds significantly if you carry recursion to more levels, actually giving O(nlogn), but with a bigger constant. So worth doing if you can't get 50-50 split, but balanced is better if you can.
- This is intuitively why Quicksort with random splitter is good – badly unbalanced splits are rare, and not instantly fatal.

A Divide & Conquer Example: Closest Pair of Points

closest pair of points: non-geometric version

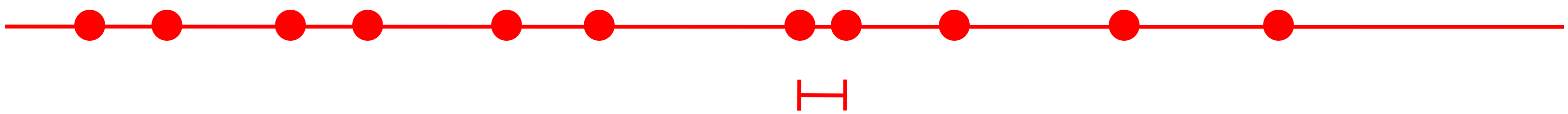
Given n points and *arbitrary* distances between them, find the closest pair. (E.g., think of distance as airfare – definitely not Euclidean distance!)



Must look at all n choose 2 pairwise distances, else any one you didn't check might be the shortest.

closest pair of points: 1 dimensional version

Given n points on the real line, find the closest pair



Closest pair is *adjacent* in ordered list

Time O(n log n) to sort, if needed

Plus O(n) to scan adjacent pairs

Key point: do *not* need to calc distances between all pairs: exploit geometry + ordering

closest pair of points: 2 dimensional version

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 $\Theta(n^2)$ time.

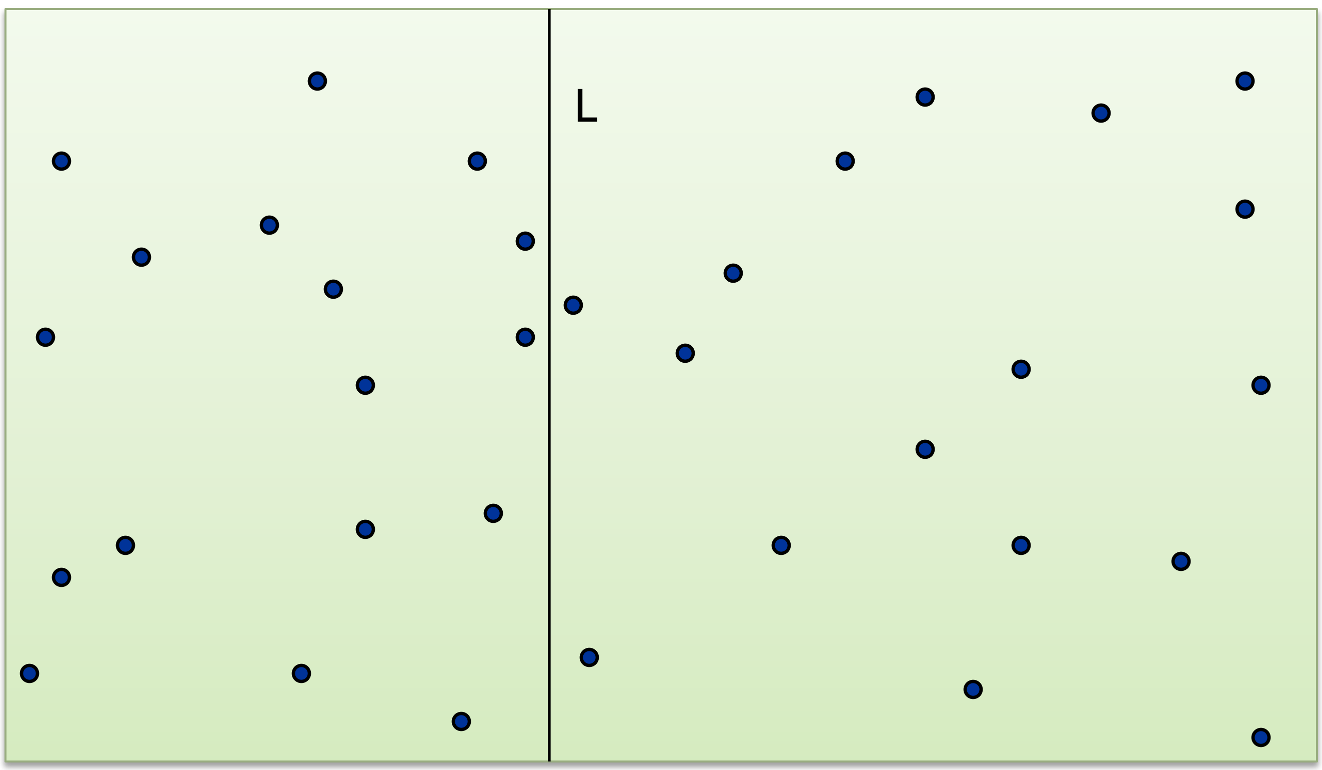
1-D version. O(n log n) easy if points are on a line.

Assumption. No two points have same x coordinate.

↑ Just to simplify presentation

Algorithm.

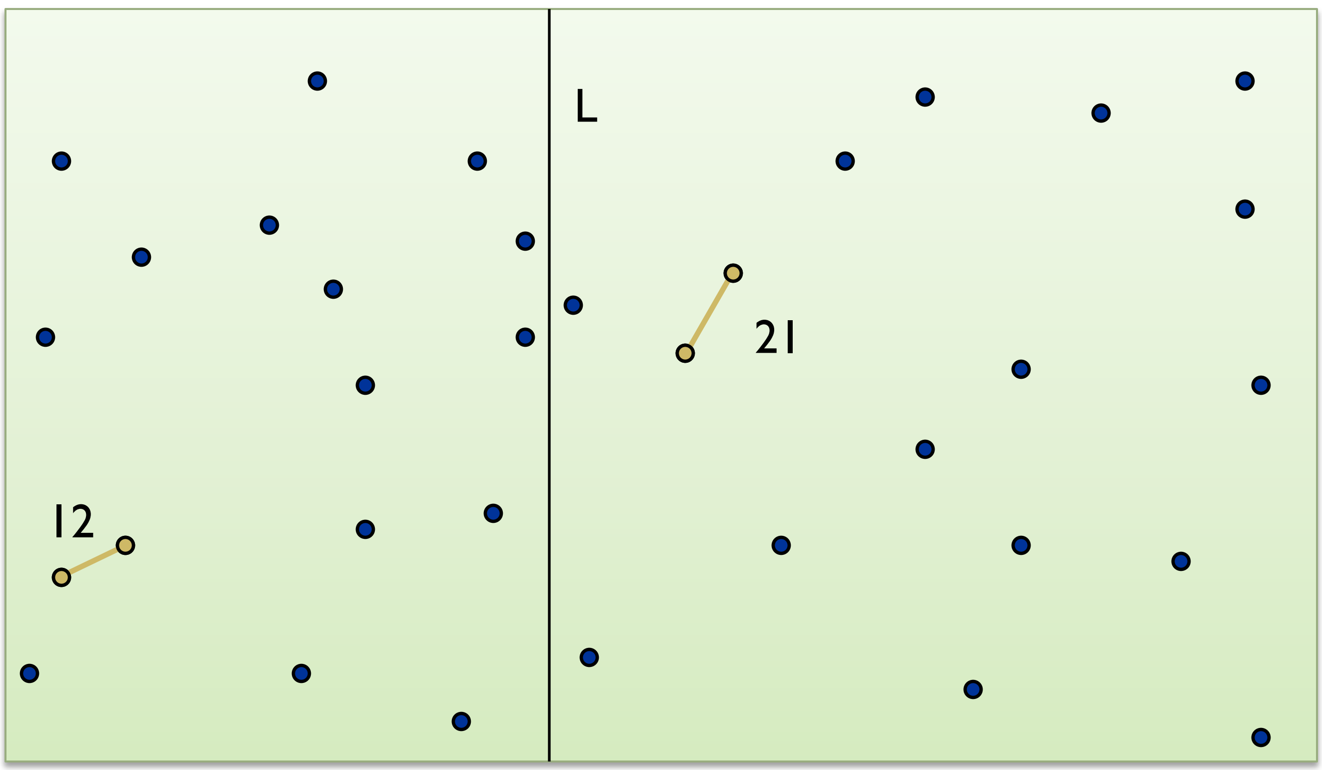
Divide: draw vertical line L with ≈ **n/2 points on** each side.



Algorithm.

Divide: draw vertical line L with ≈ **n/2 points on** each side.

Conquer: find closest pair on each side, recursively.



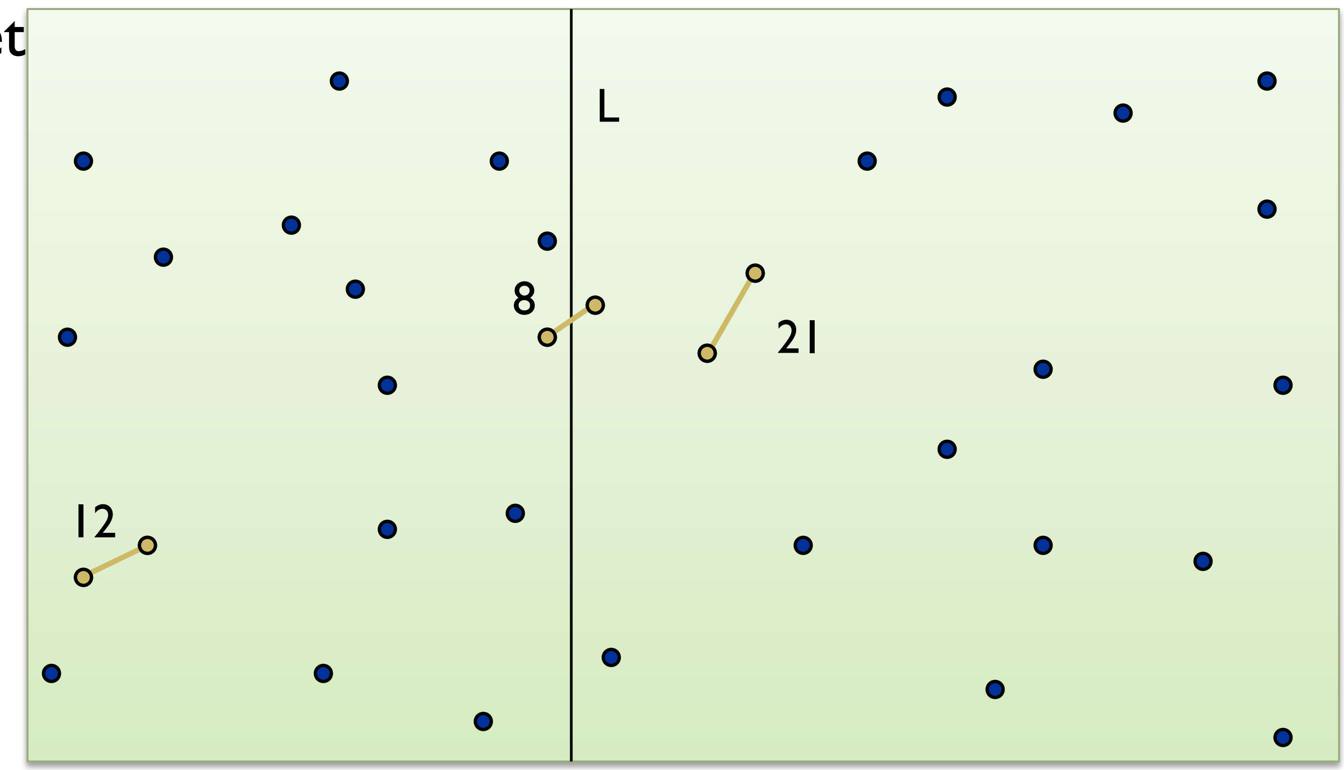
Algorithm.

Divide: draw vertical line L with ≈ **n/2 points on** each side.

Conquer: find closest pair on each side, recursively.

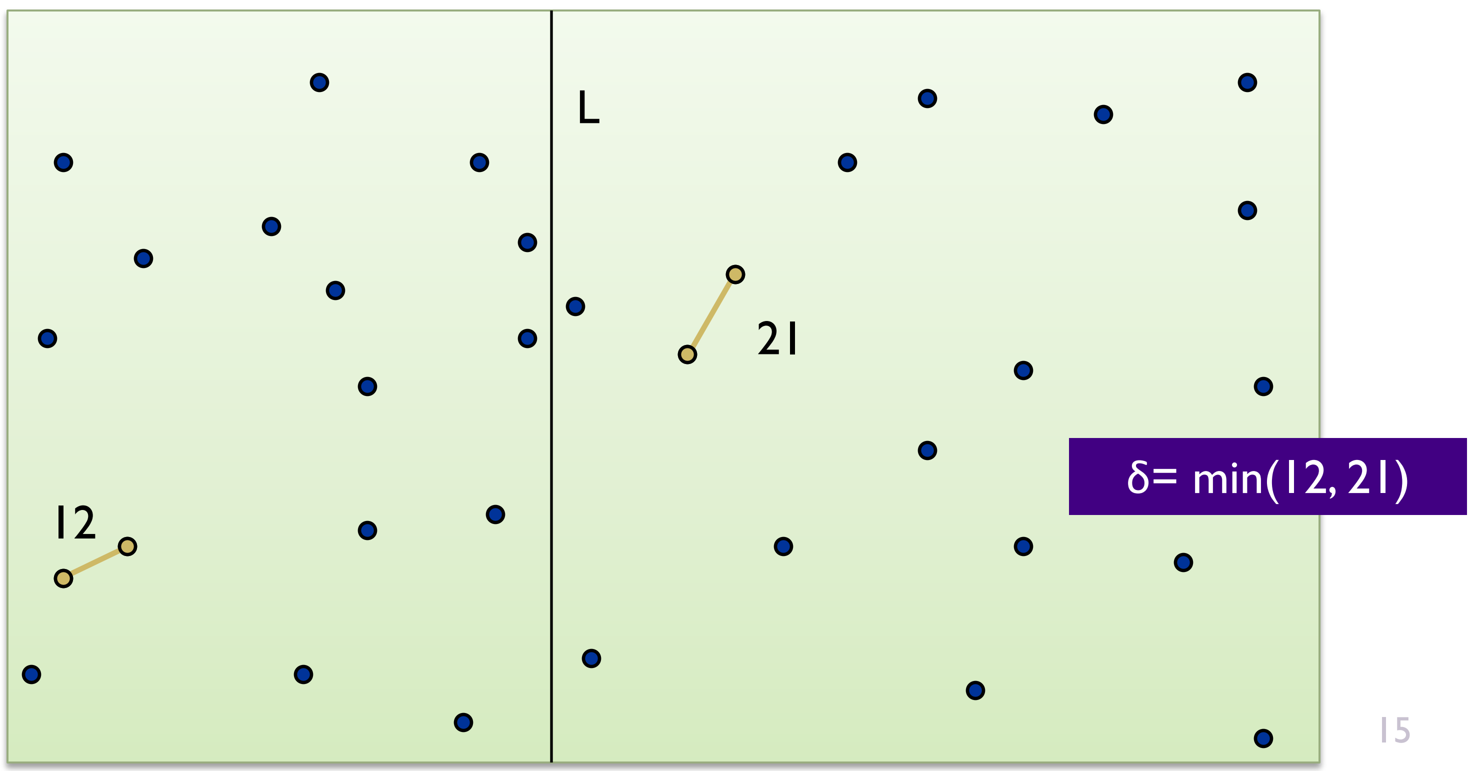
Combine to find closest pair overall

Ret



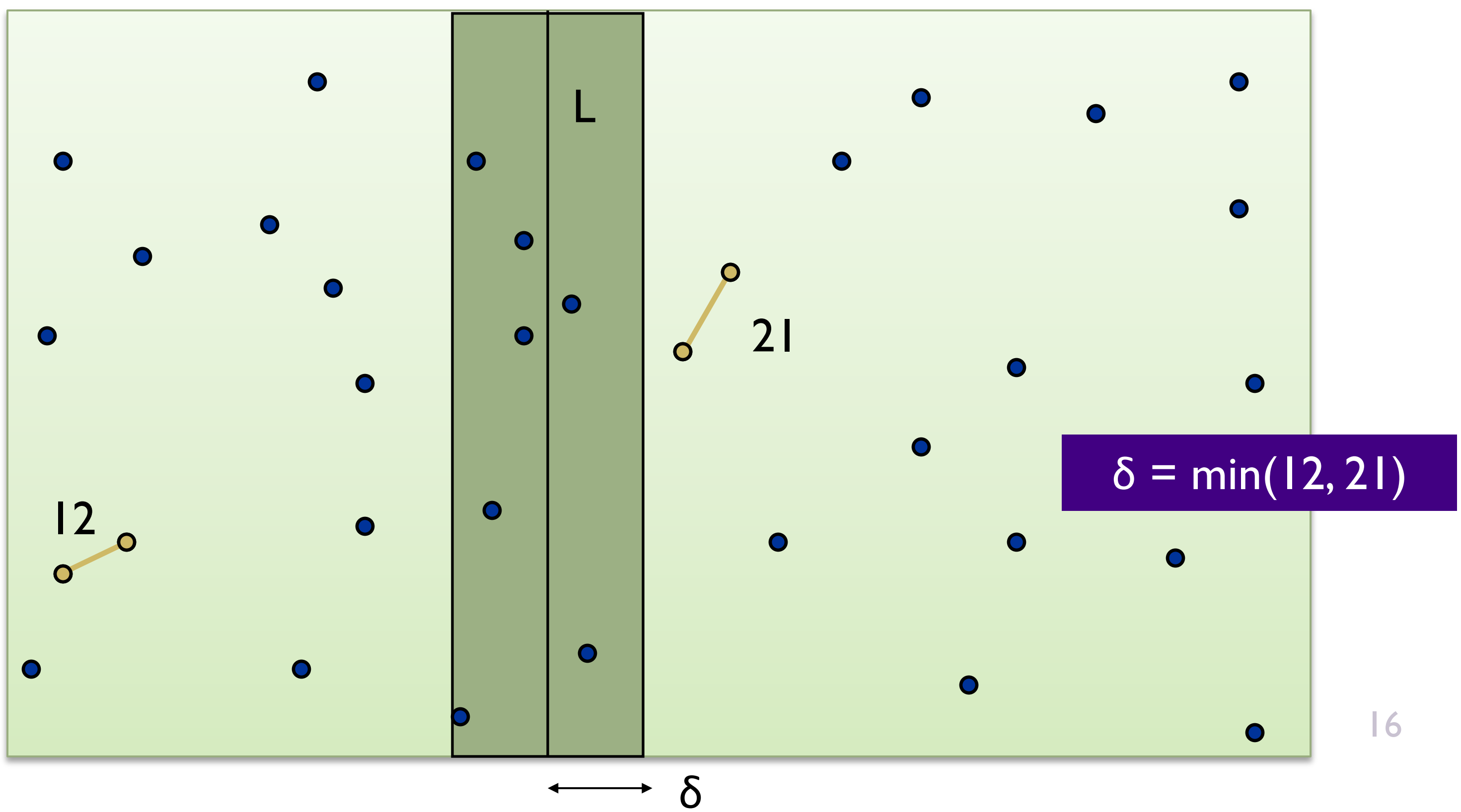
seems like $\Theta(n^2)$?

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



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

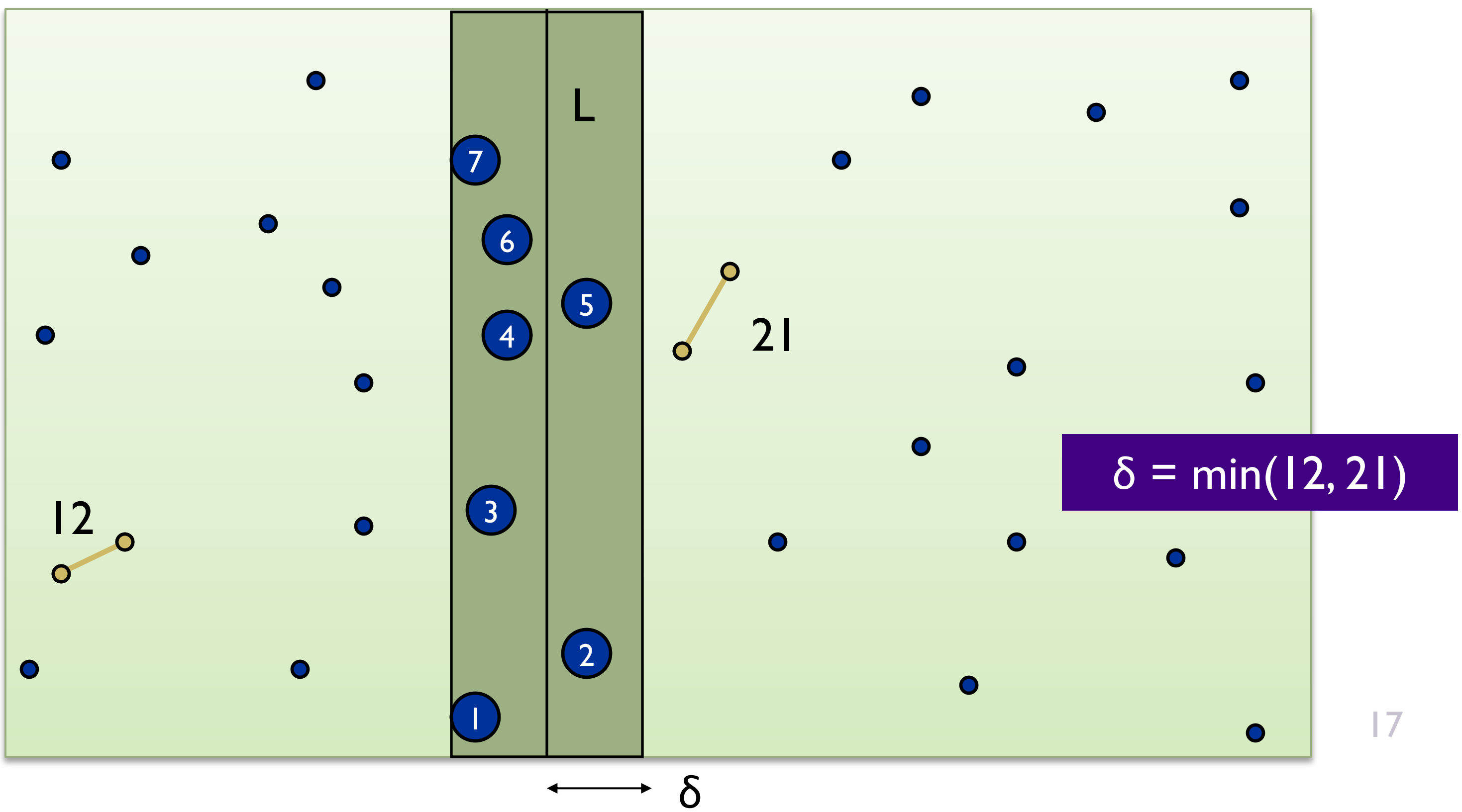
Observation: suffices to consider points within δ of line L.



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

Observation: suffices to consider points within δ of line L.

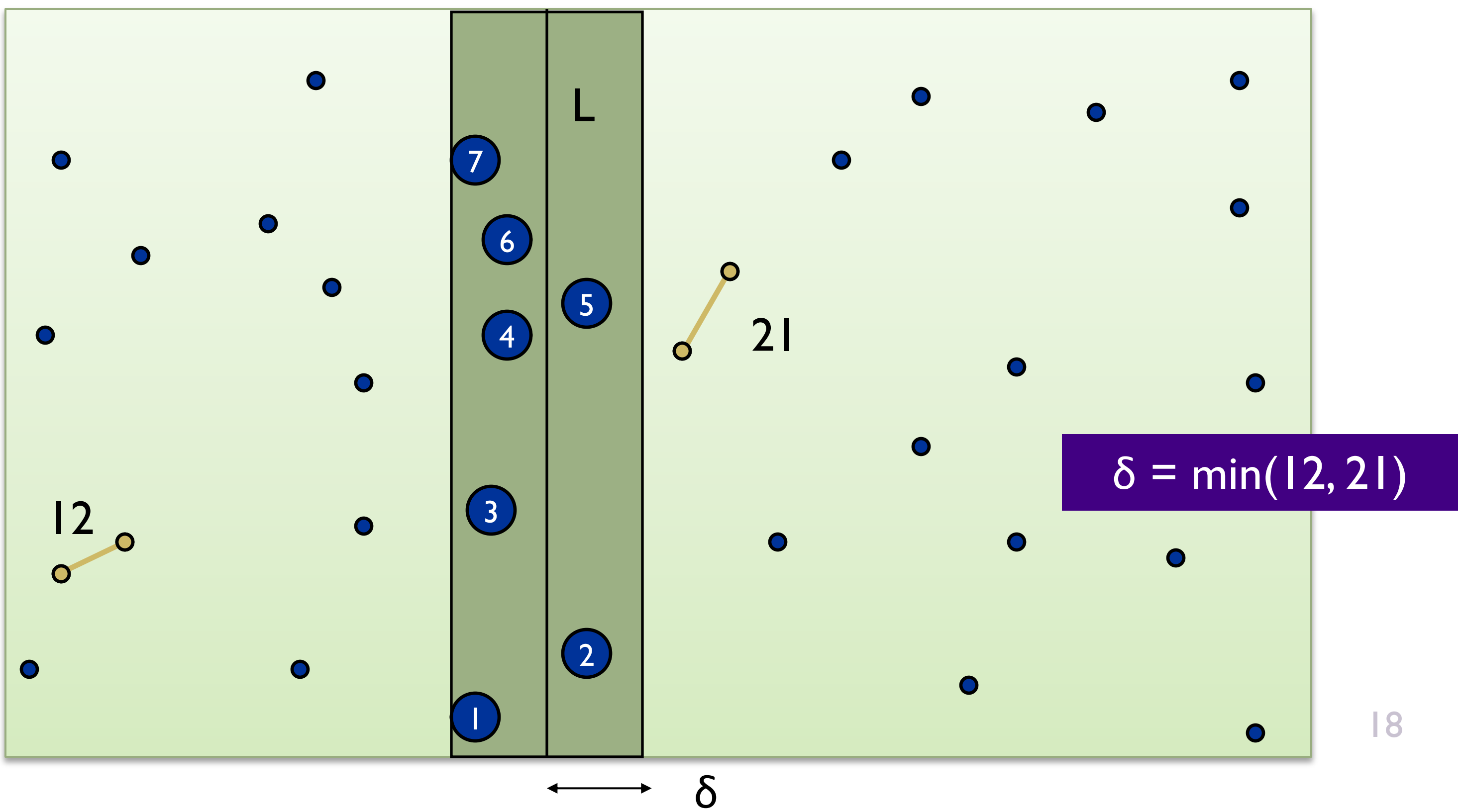
Almost the one-D problem again: Sort points in 2δ-strip by their y coordinate.



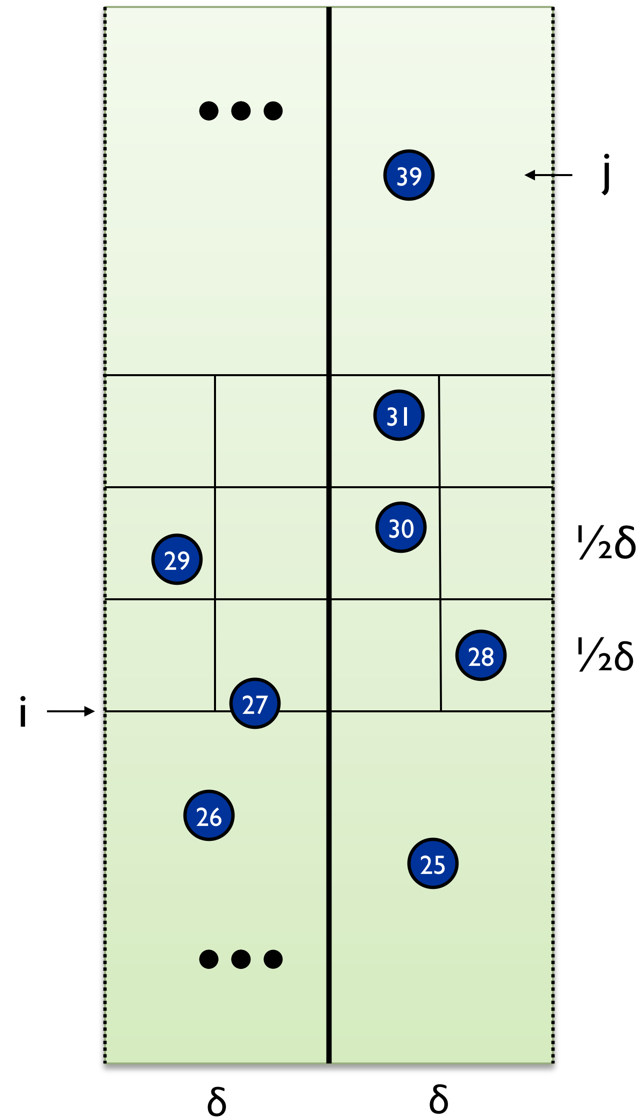
Find closest pair with one point in each side, *assuming* distance < δ.

Observation: suffices to consider points within d of line L.

Almost the one-D problem again: Sort points in 2d-strip by their y coordinate. Only check pts within 8 in sorted list!



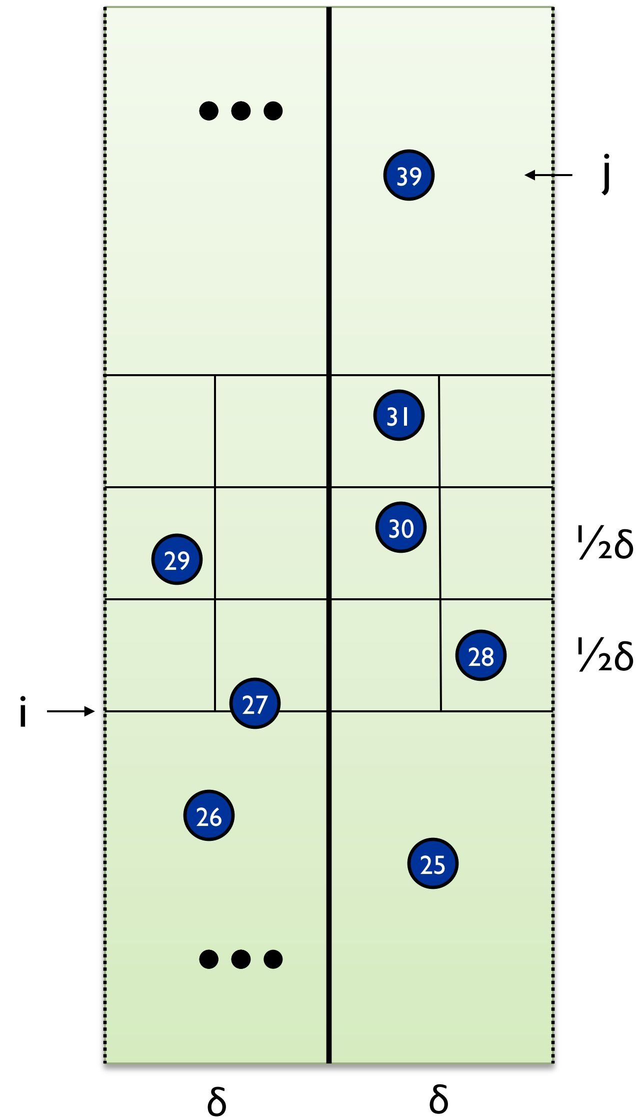
Claim: No two points lie in the same ½δ-by-½δ box.



Claim: No two points lie in the same ½δ-by-½δ box.

Pf: Such points would be within

$$\delta\sqrt{\left(\frac{1}{2}\right)^2+\left(\frac{1}{2}\right)^2}=\delta\sqrt{\frac{1}{2}}=\delta\frac{\sqrt{2}}{2}\approx 0.7\delta<\delta$$



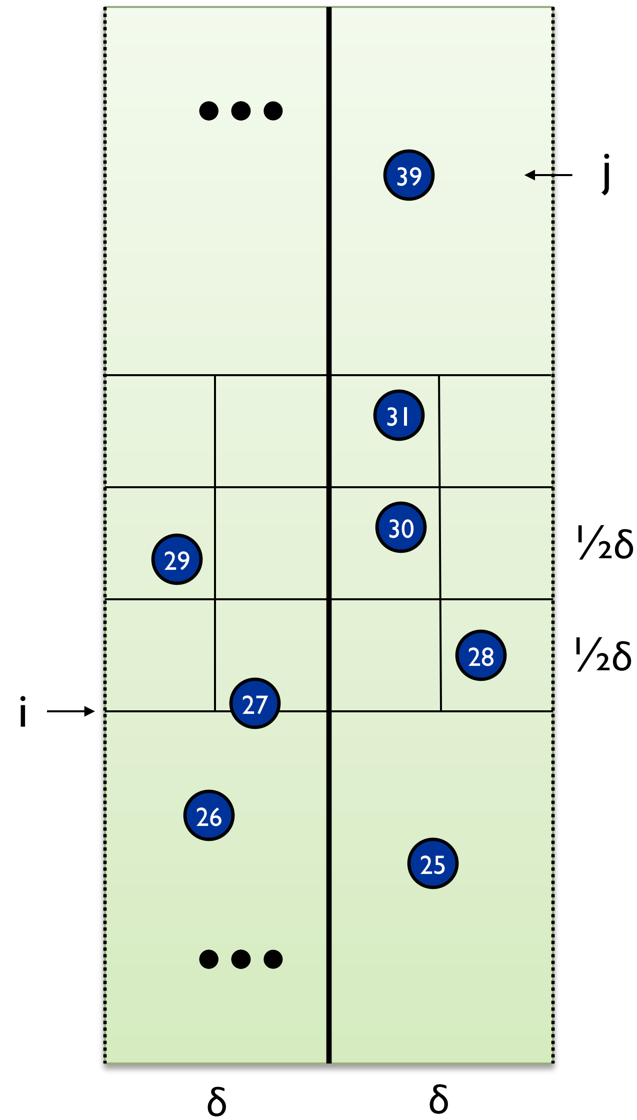
Claim: No two points lie in the same ½δ-by-½δ box.

Pf: Such points would be within

$$\delta\sqrt{\left(\frac{1}{2}\right)^2+\left(\frac{1}{2}\right)^2} = \delta\sqrt{\frac{1}{2}} = \delta\frac{\sqrt{2}}{2} \approx 0.7\delta < \delta$$

Def. Let s_i have the i^{th} smallest y-coordinate among points in the 2δ-width-strip.

Claim: If $|i - j| > 11$, then the distance between s_i and s_j is $> \delta$.



Claim: No two points lie in the same ½δ-by-½δ box.

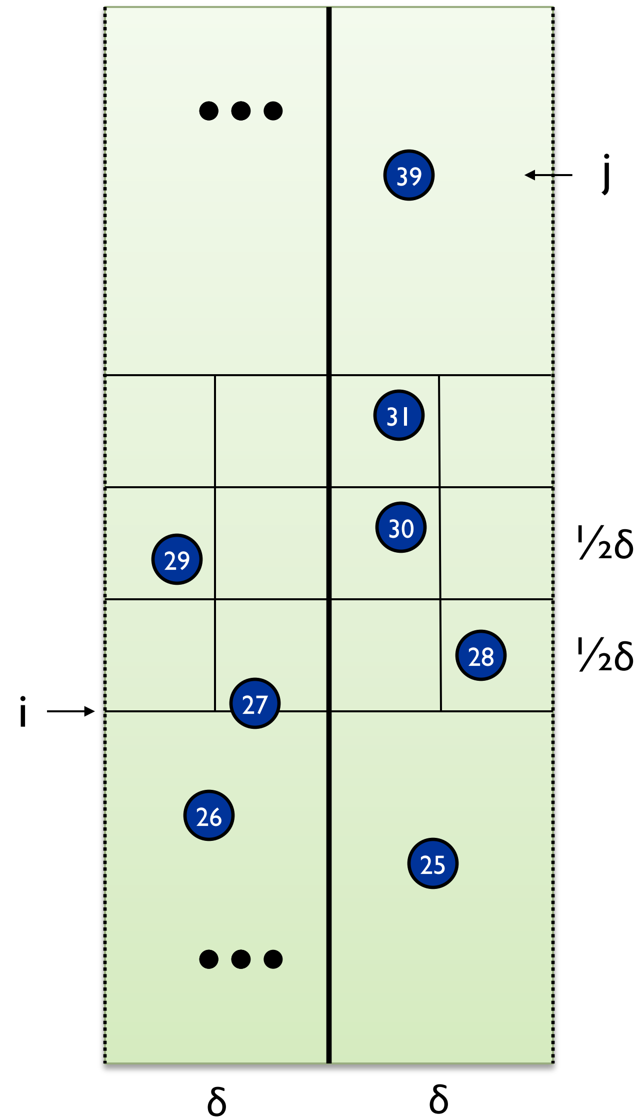
Pf: Such points would be within

$$\delta\sqrt{\left(\frac{1}{2}\right)^2+\left(\frac{1}{2}\right)^2}=\delta\sqrt{\frac{1}{2}}=\delta\frac{\sqrt{2}}{2}\approx 0.7\delta<\delta$$

Def. Let s_i have the i^{th} smallest y-coordinate among points in the 2δ-width-strip.

Claim: If |i – j| > 11, then the distance between s_i and s_j is > δ.

Pf: only 11 boxes within +δ of $y(s_i)$.



```
Closest-Pair(p1, …, pn) {
   if(n <= ??) return ??

   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 p[1]…p[m] by y-coordinate.

   for i = 1..m
      for k = 1…11
        if i+k <= m
           δ = min(δ, distance(p[i], p[i+k]));

   return δ.
}
```

Analysis, I: Let D(n) be the number of pairwise distance calculations in the Closest-Pair Algorithm when run on n>1 points

$$D(n) \le \left\{ \begin{array}{cc} 0 & n=1 \\ 2D\left(n/2\right) + 11n & n>1 \end{array} \right\} \Rightarrow D(n) = O(n \log n)$$

BUT – that's only the number of *distance calculations*

What if we counted running time?

Analysis, II: Let T(n) be the running time in the Closest-Pair Algorithm when run on n > 1 points

$$T(n) \le \begin{cases} 0 & n = 1 \\ T(n/2) + O(n \log n) & n > 1 \end{cases} \Rightarrow T(n) = O(n \log^2 n)$$

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

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

- Sort by x at top level only.
- Each recursive call returns δ *and* list of all points sorted by y
- Sort by merging two pre-sorted lists.

$$T(n) \le 2T(n/2) + O(n) \Rightarrow \mathrm{T}(n) = O(n \log n)$$

Recurrences

Applications:

multiplying numbers

multiplying matrices

computing medians

Idea:

“Two halves are better than a whole”

if the base algorithm has super-linear complexity.

“If a little's good, then more's better”

repeat above, recursively

Applications: Many.

Binary Search, Merge Sort, (Quicksort), Closest points, Integer multiply,…

Recurrences

Above: Where they come from, how to find them

Next: how to solve them

divide and conquer – master recurrence

$T(n) = aT(n/b)+cn^d$ then

$a > b^d \Rightarrow T(n) = \Theta(n^{\log_b a})$ [many subprobs → leaves dominate]

$a < b^d \Rightarrow T(n) = \Theta(n^d)$ [few subprobs → top level dominates]

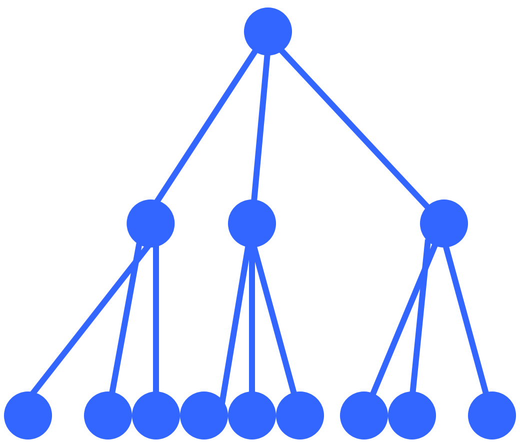
$a = b^d \Rightarrow T(n) = \Theta(n^d \log n)$ [balanced → all log n levels contribute]

Fine print:

$a \geq 1$; $b > 1$; $c, d \geq 0$; $T(1) = c$;

a, b, k, t integers.

Solve: $T(n) = a\,T(n/b) + cn^d$



Level	Num	Size	Work
0	$1 = a^0$	n	cn^d
1	a^1	n/b	$ac(n/b)^d$
2	a^2	n/b^2	$a^2c(n/b^2)^d$
…	…	…	…
i	a^i	n/b^i	$a^i\,c\,(n/b^i)^d$
…	…	…	…
k-1	a^{k-1}	n/b^{k-1}	$a^{k-1}c(n/b^{k-1})^d$
k	a^k	$n/b^k = 1$	$a^k\,T(1)$

$n = b^k$; $k = \log_b n$

Total Work: $= \sum_{i=0}^{\log_b n} a^i c(n/b^i)^d$ (add last col)

Theorem:

$$1 + x + x^2 + x^3 + \ldots + x^k = (x^{k+1}-1)/(x-1)$$

proof:

$$S = 1 + x + x^2 + x^3 + \ldots + x^k$$

$$xS = x + x^2 + x^3 + \ldots + x^k + x^{k+1}$$

$$xS-S = x^{k+1} - 1$$

$$S(x-1) = x^{k+1} - 1$$

$$S = (x^{k+1}-1)/(x-1)$$

$$T(1) = d$$
$$T(n) = a\,T(n/b) + cn^d \quad , a > b^d$$

$$T(n) = \sum_{i=0}^{\log_b n} a^i c(n/b^i)^d$$

$$= cn^d \sum_{i=0}^{\log_b n} (a/b^d)^i$$

$$= cn^d \frac{\left(\frac{a}{b^d}\right)^{\log_b n+1} - 1}{\left(\frac{a}{b^d}\right) - 1}$$

$$\sum_{i=0}^{k} x^i = \frac{x^{k+1} - 1}{x - 1}$$
$$(x \neq 1)$$

Solve: T(1) = d

T(n) = a T(n/b) + cn^d , a > b^d

$$cn^d \frac{\left(\frac{a}{b^d}\right)^{\log_b n+1} - 1}{\left(\frac{a}{b^d}\right) - 1} < cn^d \frac{\left(\frac{a}{b^d}\right)^{\log_b n+1}}{\left(\frac{a}{b^d}\right) - 1}$$

$$= c\left(\frac{n^d}{b^{d\log_b n}}\right)\left(\frac{a}{b^d}\right)\frac{a^{\log_b n}}{\left(\frac{a}{b^d}\right) - 1}$$

$$= c\left(\frac{a}{b^d}\right)\frac{a^{\log_b n}}{\left(\frac{a}{b^d}\right) - 1}$$

$$= O(n^{\log_b a})$$

$$n^d = \left(b^{\log_b n}\right)^d = b^{d\log_b n}$$

$$a^{\log_b n} = \left(b^{\log_b a}\right)^{\log_b n} = \left(b^{\log_b n}\right)^{\log_b a} = n^{\log_b a}$$

Solve: $T(1) = d$

$T(n) = a\,T(n/b) + cn^d \quad , a < b^d$

$$T(n) = \sum_{i=0}^{\log_b n} a^i c(n/b^i)^d$$

$$= cn^d \sum_{i=0}^{\log_b n} a^i / b^{id}$$

$$= cn^d \frac{1-\left(\frac{a}{b^d}\right)^{\log_b n+1}}{1-\left(\frac{a}{b^d}\right)}$$

$$< cn^d \frac{1}{1-\left(\frac{a}{b^d}\right)}$$

$$= O(n^d)$$

$$\sum_{i=0}^{k} x^i = \frac{x^{k+1}-1}{x-1}$$

$$(x \neq 1)$$

Solve: $T(1) = d$

$T(n) = a\,T(n/b) + cn^d \quad , a = b^d$

$$T(n) = \sum_{i=0}^{\log_b n} a^i c(n/b^i)^d$$

$$= cn^d \sum_{i=0}^{\log_b n} a^i / b^{id}$$

$$= O(n^d \log_b n)$$

$T(n) = aT(n/b)+cn^d$ for $n > b$ then

$a > b^d \Rightarrow T(n) = \Theta(n^{\log_b a})$ [many subprobs → leaves dominate]

$a < b^d \Rightarrow T(n) = \Theta(n^d)$ [few subprobs → top level dominates]

$a = b^d \Rightarrow T(n) = \Theta(n^d \log n)$ [balanced → all log n levels contribute]

Fine print:

$a \geq 1$; $b > 1$; $c, d \geq 0$; $T(1) = c$;

a, b, k, t integers.

Integer Multiplication

Add. Given two n-bit integers a and b, compute a + b.

Add

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

O(n) bit operations.

Add. Given two n-bit integers a and b, compute a + b.

O(n) bit operations.

Add

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

Multiply. Given two n-bit integers a and b, compute a × b.

The "grade school" method:

Multiply

								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	
						1	1	0	1	0	1	0	1		
					1	1	0	1	0	1	0	1			
				1	1	0	1	0	1	0	1				
			1	1	0	1	0	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	0	0	0	0	0	0	0	0	0	1

Add. Given two n-bit integers a and b, compute a + b.

O(n) bit operations.

Add

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

Multiply. Given two n-bit integers a and b, compute a × b.
The "grade school" method:

$\Theta(n^2)$ bit operations.

Multiply

								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	
						1	1	0	1	0	1	0	1		
					1	1	0	1	0	1	0	1			
				1	1	0	1	0	1	0	1				
			1	1	0	1	0	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	0	0	0	0	0	0	0	0	0	1

divide & conquer multiplication: warmup

To multiply two 2-digit integers:

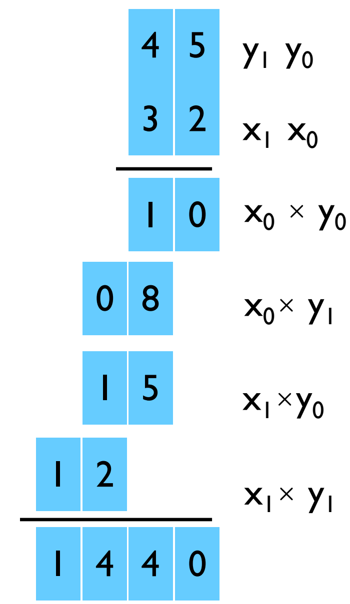
Multiply four 1-digit integers.

Add, shift some 2-digit integers to obtain result.

$$
\begin{aligned}
x &= 10 \cdot x_1 + x_0 \\
y &= 10 \cdot y_1 + y_0 \\
xy &= \left(10 \cdot x_1 + x_0\right)\left(10 \cdot y_1 + y_0\right) \\
&= 100 \cdot x_1 y_1 + 10 \cdot \left(x_1 y_0 + x_0 y_1\right) + x_0 y_0
\end{aligned}
$$

Same idea works for *long* integers –

can split them into 4 half-sized ints

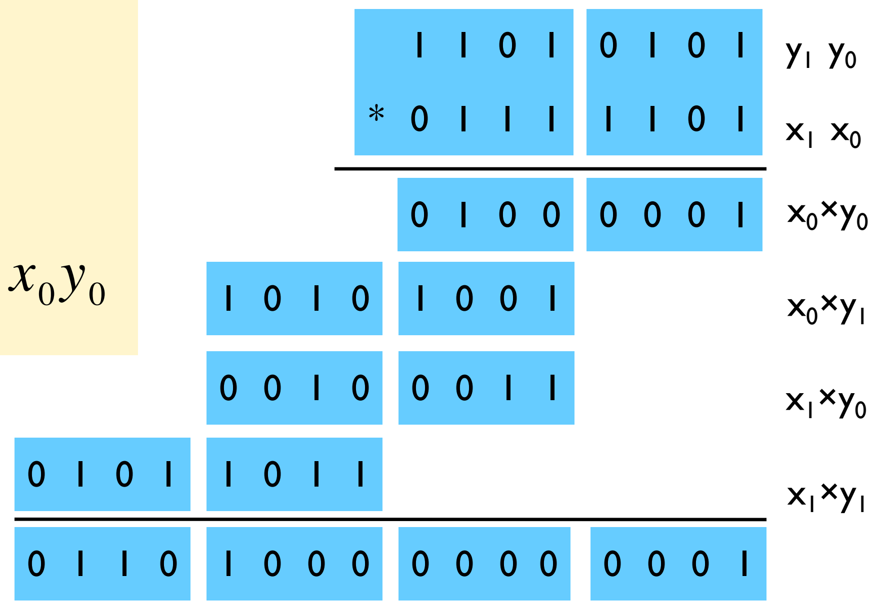


To multiply two n-bit integers:

Multiply four ½n-bit integers.

Add two ½n-bit integers, and shift to obtain result.

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right) \\
&= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot \left(x_1 y_0 + x_0 y_1\right) + x_0 y_0
\end{aligned}
$$



$$
T(n) = \underbrace{4T\left(n/2\right)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}}
$$

To multiply two n-bit integers:

Multiply four ½n-bit integers.

Add two ½n-bit integers, and shift to obtain result.

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right) \\
&= 2^{n} \cdot x_1 y_1 + 2^{n/2} \cdot \left(x_1 y_0 + x_0 y_1\right) + x_0 y_0
\end{aligned}
$$

$$
\mathrm{T}(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow \mathrm{T}(n) = \Theta(n^2)
$$

		1 1 0 1	0 1 0 1	y_1 y_0
		* 0 1 1 1	1 1 0 1	x_1 x_0
		0 1 0 0	0 0 0 1	$x_0 \times y_0$
	1 0 1 0	1 0 0 1		$x_0 \times y_1$
	0 0 1 0	0 0 1 1		$x_1 \times y_0$
0 1 0 1	1 0 1 1			$x_1 \times y_1$
0 1 1 0	1 0 0 0	0 0 0 0	0 0 0 1	

key trick: 2 multiplies for the price of 1:

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right) \\
&= 2^{n} \cdot x_1 y_1 + 2^{n/2} \cdot \left(x_1 y_0 + x_0 y_1\right) + x_0 y_0
\end{aligned}
$$

Well, ok, 4 for 3 is more accurate…

$$
\begin{aligned}
\alpha &= x_1 + x_0 \\
\beta &= y_1 + y_0 \\
\alpha\beta &= \left(x_1 + x_0\right)\left(y_1 + y_0\right) \\
&= x_1 y_1 + \left(x_1 y_0 + x_0 y_1\right) + x_0 y_0 \\
\left(x_1 y_0 + x_0 y_1\right) &= \alpha\beta - x_1 y_1 - x_0 y_0
\end{aligned}
$$

To multiply two n-bit integers:

Add two ½n bit integers.

Multiply three ½n-bit integers.

Add, subtract, and shift ½n-bit integers to obtain result.

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= 2^{n} \cdot x_1 y_1 + 2^{n/2} \cdot \left(x_1 y_0 + x_0 y_1\right) + x_0 y_0 \\
&= 2^{n} \cdot \underbrace{x_1 y_1}_{A} + 2^{n/2} \cdot \left(\underbrace{(x_1 + x_0)(y_1 + y_0)}_{B} - \underbrace{x_1 y_1}_{A} - \underbrace{x_0 y_0}_{C} \right) + \underbrace{x_0 y_0}_{C}
\end{aligned}
$$

Theorem. [Karatsuba-Ofman, 1962] Can multiply two n-digit integers in $O(n^{1.585})$ bit operations.

$$
\begin{aligned}
T(n) &\le 3T(n/2) + O(n) \\
\Rightarrow \mathrm{T}(n) &= O(n^{\log_2 3}) = O(n^{1.585})
\end{aligned}
$$

Naïve: $\Theta(n^2)$

Karatsuba: $\Theta(n^{1.59\ldots})$

Amusing exercise: generalize Karatsuba to do 5 size n/3 subproblems $\rightarrow \Theta(n^{1.46\ldots})$

Best known: $\Theta(n \log n \log\log n)$

"Fast Fourier Transform”

Another Example:

Matrix Multiplication –

Strassen’s Method

Multiplying Matrices

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \bullet \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} + a_{14}b_{42} & \circ & a_{11}b_{14} + a_{12}b_{24} + a_{13}b_{34} + a_{14}b_{44} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} + a_{24}b_{41} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42} & \circ & a_{21}b_{14} + a_{22}b_{24} + a_{23}b_{34} + a_{24}b_{44} \\ a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} + a_{34}b_{41} & a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} + a_{34}b_{42} & \circ & a_{31}b_{14} + a_{32}b_{24} + a_{33}b_{34} + a_{34}b_{44} \\ a_{41}b_{11} + a_{42}b_{21} + a_{43}b_{31} + a_{44}b_{41} & a_{41}b_{12} + a_{42}b_{22} + a_{43}b_{32} + a_{44}b_{42} & \circ & a_{41}b_{14} + a_{42}b_{24} + a_{43}b_{34} + a_{44}b_{44} \end{bmatrix}$$

n^3 multiplications, n^3-n^2 additions

```
for i = 1 to n
    for j = 1 to n
        C[i,j] = 0
        for k = 1 to n
            C[i,j] = C[i,j] + A[i,k] * B[k,j]
```

n^3 multiplications, n^3-n^2 additions

Multiplying Matrices

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \bullet \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} + a_{14}b_{42} & \circ & a_{11}b_{14} + a_{12}b_{24} + a_{13}b_{34} + a_{14}b_{44} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} + a_{24}b_{41} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42} & \circ & a_{21}b_{14} + a_{22}b_{24} + a_{23}b_{34} + a_{24}b_{44} \\ a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} + a_{34}b_{41} & a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} + a_{34}b_{42} & \circ & a_{31}b_{14} + a_{32}b_{24} + a_{33}b_{34} + a_{34}b_{44} \\ a_{41}b_{11} + a_{42}b_{21} + a_{43}b_{31} + a_{44}b_{41} & a_{41}b_{12} + a_{42}b_{22} + a_{43}b_{32} + a_{44}b_{42} & \circ & a_{41}b_{14} + a_{42}b_{24} + a_{43}b_{34} + a_{44}b_{44} \end{bmatrix}$$

Multiplying Matrices

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \bullet \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} + a_{14}b_{42} & \circ & a_{11}b_{14} + a_{12}b_{24} + a_{13}b_{34} + a_{14}b_{44} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} + a_{24}b_{41} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42} & \circ & a_{21}b_{14} + a_{22}b_{24} + a_{23}b_{34} + a_{24}b_{44} \\ a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} + a_{34}b_{41} & a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} + a_{34}b_{42} & \circ & a_{31}b_{14} + a_{32}b_{24} + a_{33}b_{34} + a_{34}b_{44} \\ a_{41}b_{11} + a_{42}b_{21} + a_{43}b_{31} + a_{44}b_{41} & a_{41}b_{12} + a_{42}b_{22} + a_{43}b_{32} + a_{44}b_{42} & \circ & a_{41}b_{14} + a_{42}b_{24} + a_{43}b_{34} + a_{44}b_{44} \end{bmatrix}$$

$$\left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] \bullet \left[\begin{array}{cc|cc} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ \hline b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{array}\right]$$

$$\left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] \bullet \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right]$$

$$= \left[\begin{array}{cc|cc} a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31}+a_{14}b_{41} & a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32}+a_{14}b_{42} & \circ & a_{11}b_{14}+a_{12}b_{24}+a_{13}b_{34}+a_{14}b_{44} \\ a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31}+a_{24}b_{41} & a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32}+a_{24}b_{42} & \circ & a_{21}b_{14}+a_{22}b_{24}+a_{23}b_{34}+a_{24}b_{44} \\ \hline a_{31}b_{11}+a_{32}b_{21}+a_{33}b_{31}+a_{34}b_{41} & a_{31}b_{12}+a_{32}b_{22}+a_{33}b_{32}+a_{34}b_{42} & \circ & a_{31}b_{14}+a_{32}b_{24}+a_{33}b_{34}+a_{34}b_{44} \\ a_{41}b_{11}+a_{42}b_{21}+a_{43}b_{31}+a_{44}b_{41} & a_{41}b_{12}+a_{42}b_{22}+a_{43}b_{32}+a_{44}b_{42} & \circ & a_{41}b_{14}+a_{42}b_{24}+a_{43}b_{34}+a_{44}b_{44} \end{array}\right]$$

$$= \left[\begin{array}{c|c} A_{11}B_{11}+A_{12}B_{21} & A_{11}B_{12}+A_{12}B_{22} \\ \hline A_{21}B_{11}+A_{22}B_{21} & A_{21}B_{12}+A_{22}B_{22} \end{array}\right]$$

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$= \begin{pmatrix} A_{11}B_{11}+A_{12}B_{21} & A_{11}B_{12}+A_{12}B_{22} \\ A_{21}B_{11}+A_{22}B_{21} & A_{21}B_{12}+A_{22}B_{22} \end{pmatrix}$$

Counting arithmetic operations:

$T(n) = 8T(n/2) + 4(n/2)^2 = 8T(n/2) + n^2$

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 8T(n/2) + n^2 & \text{if } n > 1 \end{cases}$$

By Master Recurrence, if

$T(n) = aT(n/b)+cn^d$ & $a > b^d$ then

$T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 8}) = \Theta(n^3)$

The algorithm

$P_1 = A_{12}(B_{11}+B_{21})$

$P_2 = A_{21}(B_{12}+B_{22})$

$P_3 = (A_{11} - A_{12})B_{11}$

$P_4 = (A_{22} - A_{21})B_{22}$

$P_5 = (A_{22} - A_{12})(B_{21} - B_{22})$

$P_6 = (A_{11} - A_{21})(B_{12} - B_{11})$

$P_7 = (A_{21} - A_{12})(B_{11}+B_{22})$

$C_{11} = P_1+P_3$

$C_{12} = P_2+P_3+P_6-P_7$

$C_{21} = P_1+P_4+P_5+P_7$

$C_{22} = P_2+P_4$

Strassen's algorithm

Multiply 2x2 matrices using **7** instead of **8** multiplications
(and lots more than 4 additions)

$T(n)=7\,T(n/2)+cn^2$

$7>2^2$ so $T(n)$ is $\Theta(n^{\log_2 7})$ which is $O(n^{2.81})$

Fastest algorithms theoretically use $O(n^{2.376})$ time