

Divide and Conquer

- Mergesort
- Closest Pair
- Integer Multiplication
- Matrix Multiplication
- Maximum Sum Subarray

Divide-and-Conquer

Divide-and-conquer.

- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage.

- Break up problem of size n into **two** equal parts of size $\frac{1}{2}n$.
- Solve two parts recursively.
- Combine two solutions into overall solution in **linear time**.

Consequence.

- Brute force: n^2 .
- Divide-and-conquer: $n \log n$.

Mergesort

Sorting

Sorting. Given n elements, rearrange in ascending order.

Applications.

- Sort a list of names.
- Organize an MP3 library.
- Find the median.
- Find the closest pair.
- Binary search in a database.
- Find duplicates in a mailing list.
- ...

Mergesort

Mergesort.

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

A L G O R I T H M S

A L G O R

I T H M S

divide $O(1)$

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 $O(n)$

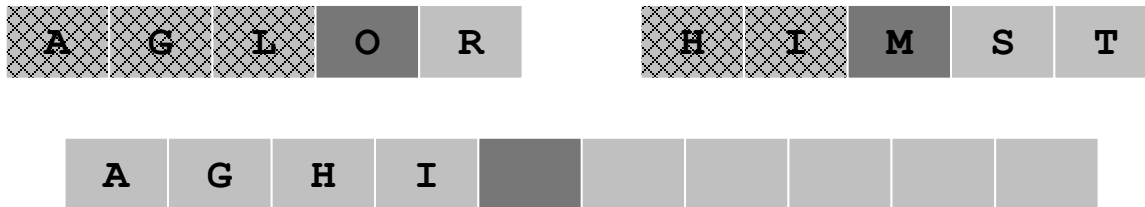
Merging

Merging. Combine two pre-sorted lists into a sorted whole.

How to merge efficiently?



- Linear number of comparisons.
- Use temporary array.



Challenging version. In-place merge

↑
using only a constant amount of extra storage

A Useful Recurrence Relation

Def. $T(n)$ = number of comparisons to mergesort an input of size n .

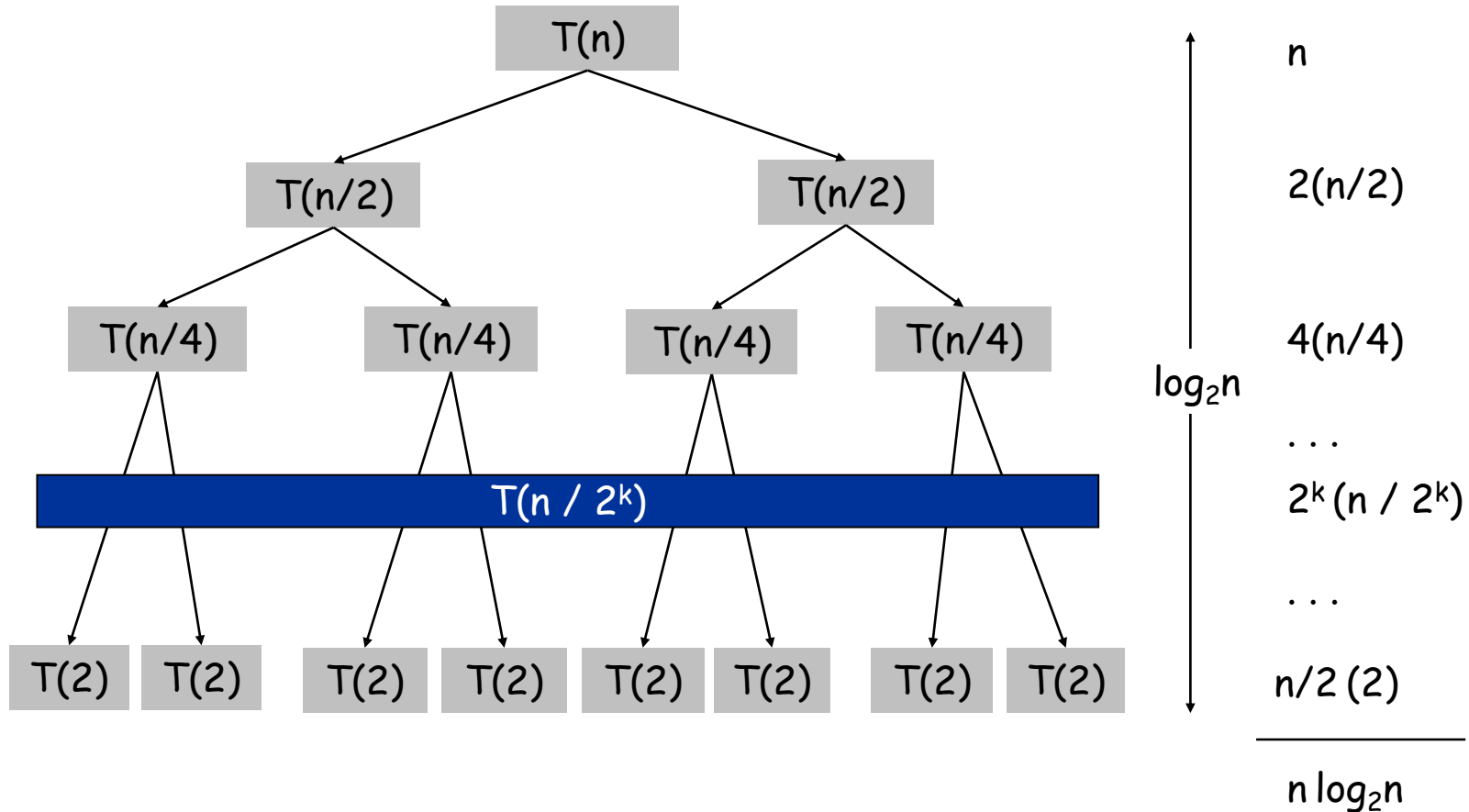
Mergesort recurrence.

$$T(n) \leq \begin{cases} 0 & \text{if } n=1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Solution. $T(n) = O(n \log_2 n)$.

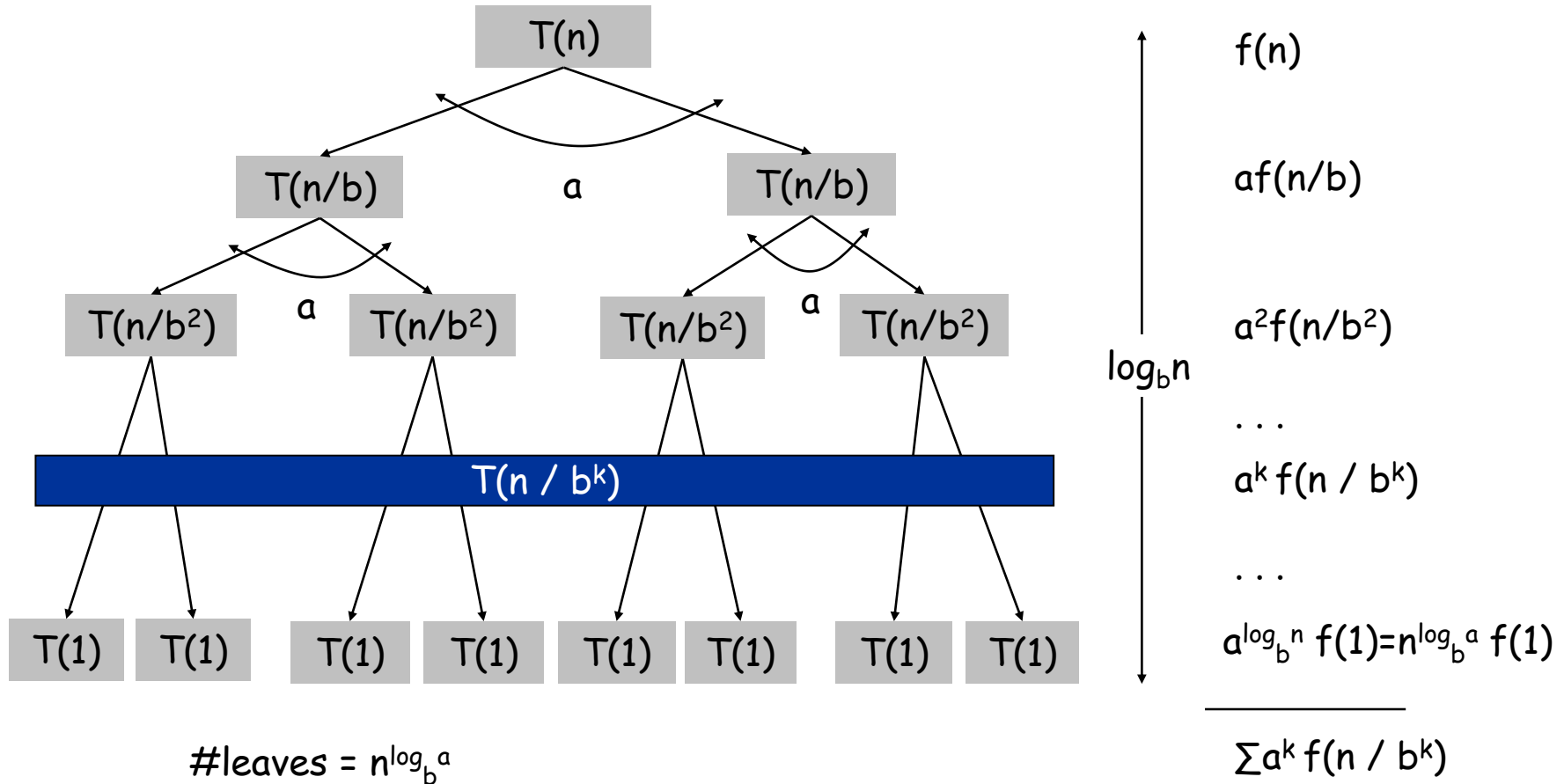
Proof by Recursion Tree

$$T(n) = \begin{cases} 0 & \text{if } n=1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$



Recursion Tree

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{aT(n/b)}_{\text{sub-problems}} + \underbrace{f(n)}_{\text{merging}} & \text{otherwise} \end{cases}$$



Master Theorem

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{aT(n/b)}_{\text{sub-problems}} + \underbrace{f(n)}_{\text{merging}} & \text{otherwise} \end{cases}$$

You can imagine above as a recursive function which calls itself: a times, each with an input of size n/b , and merge their outputs in $f(n)$ time.

Fighting between #leaves and $f(n)$

- If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.
- If $f(n)$ polynomially greater than $n^{\log_b a}$, then $T(n) = \Theta(f(n))$
- If $n^{\log_b a}$ polynomially greater than $f(n)$, then $T(n) = \Theta(n^{\log_b a})$

Note. The total input injecting to sub-problems is $(a/b)n$. Then if a/b is smaller, your running time is better.

Mergesort

What happen if we divide the array into more subproblems?.

- We have to find the minimum among a numbers in the merging step.
- So,

$$T(n)=aT(n/a)+an$$

- It is easy to see $T(n)$ is minimum when $a=2$

5.4 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.

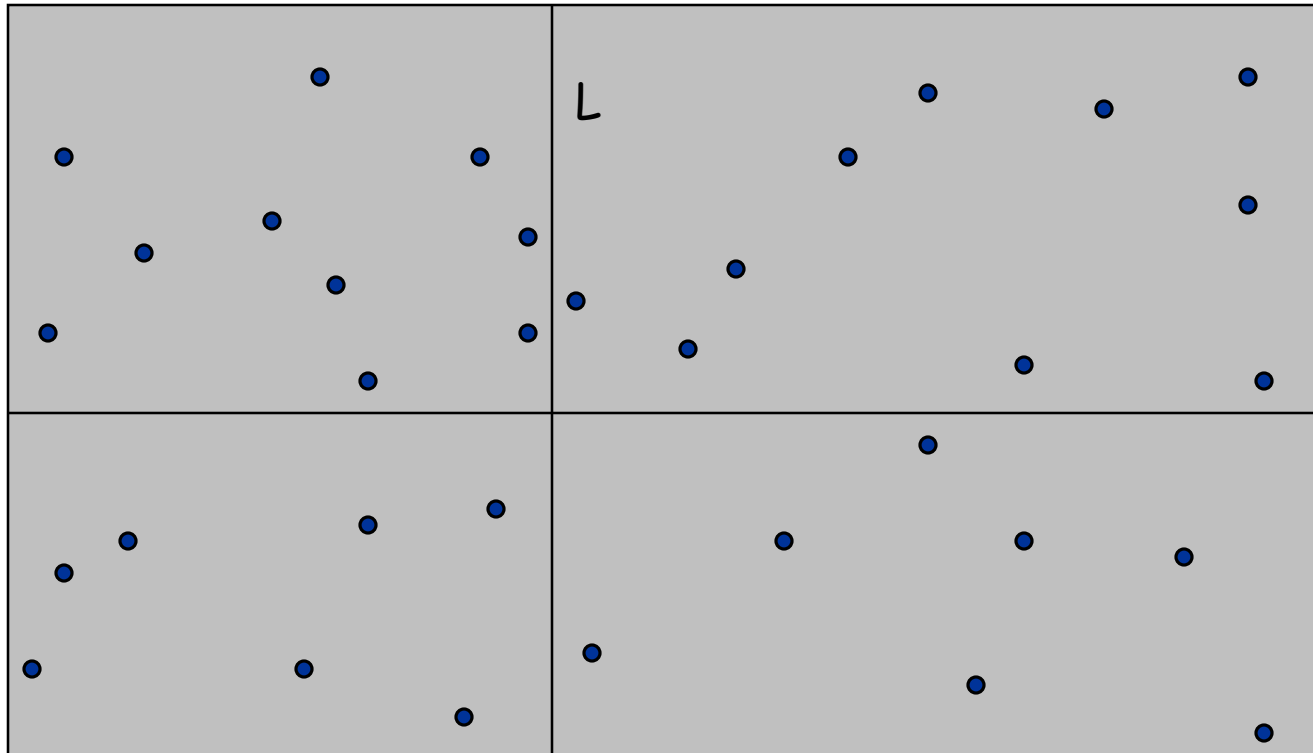
1-D version. $O(n \log n)$ easy if points are on a line.

Assumption. No two points have same x coordinate.

↑
to make presentation cleaner

Closest Pair of Points: First Attempt

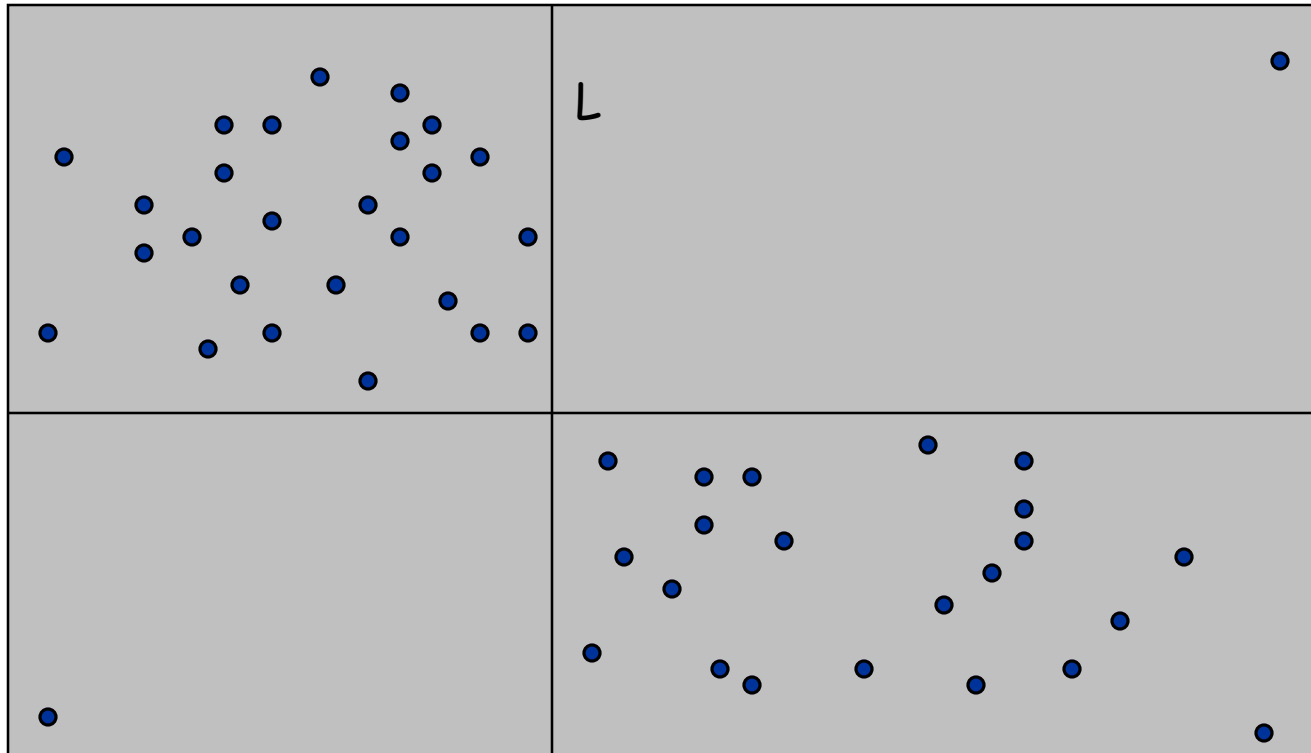
Divide. Sub-divide region into 4 quadrants.



Closest Pair of Points: First Attempt

Divide. Sub-divide region into 4 quadrants.

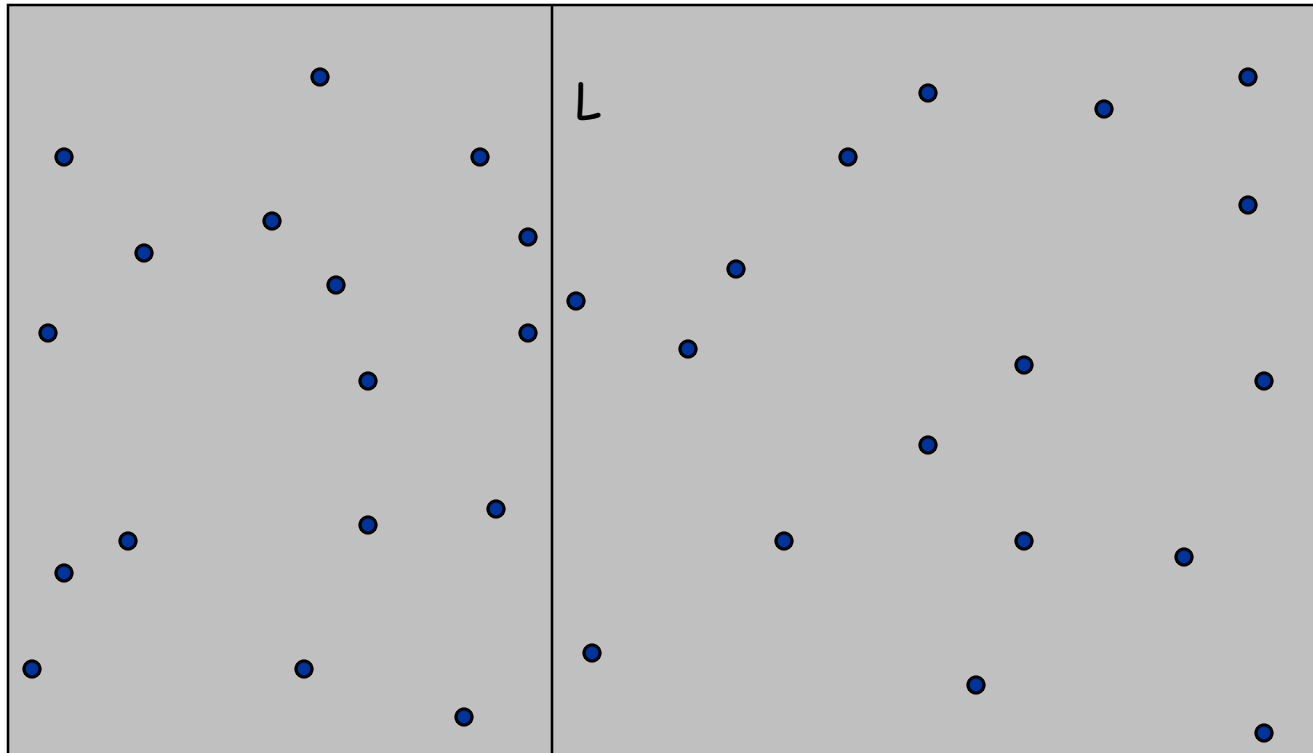
Obstacle. Impossible to ensure $n/4$ points in each piece.



Closest Pair of Points

Algorithm.

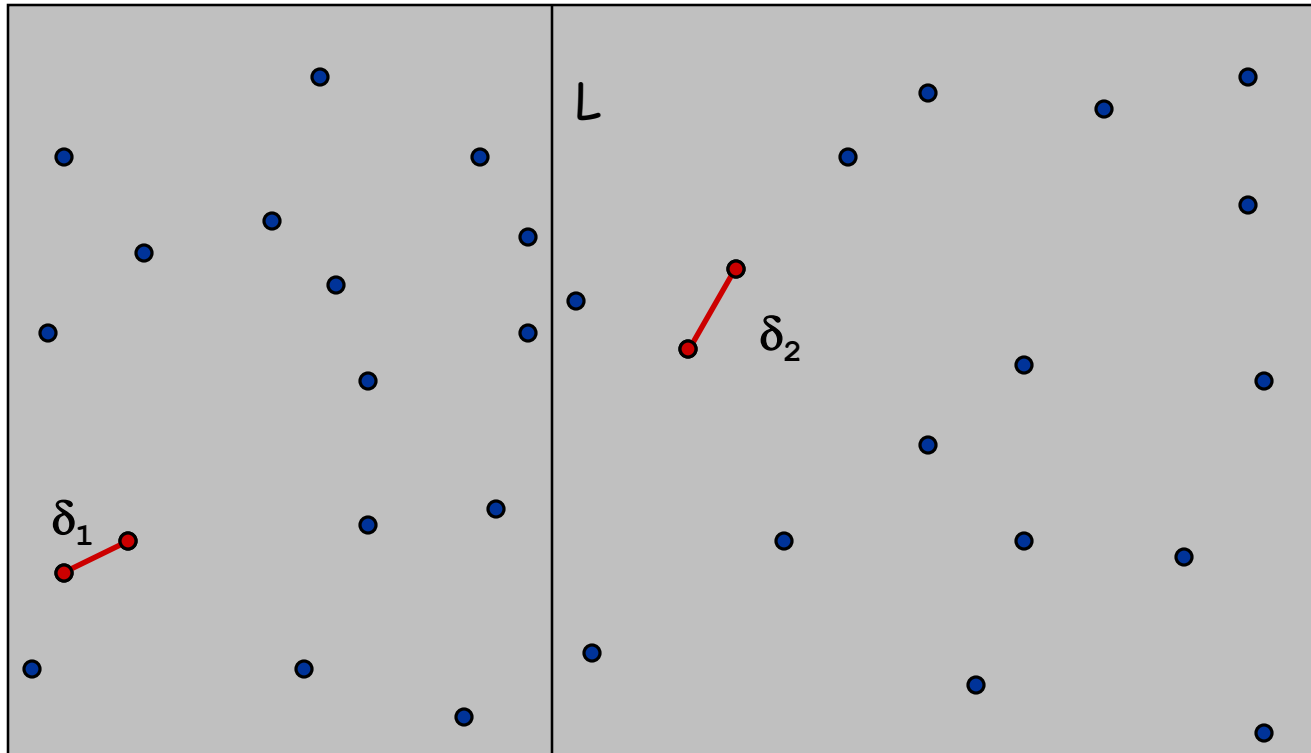
- **Divide:** draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.



Closest Pair of Points

Algorithm.

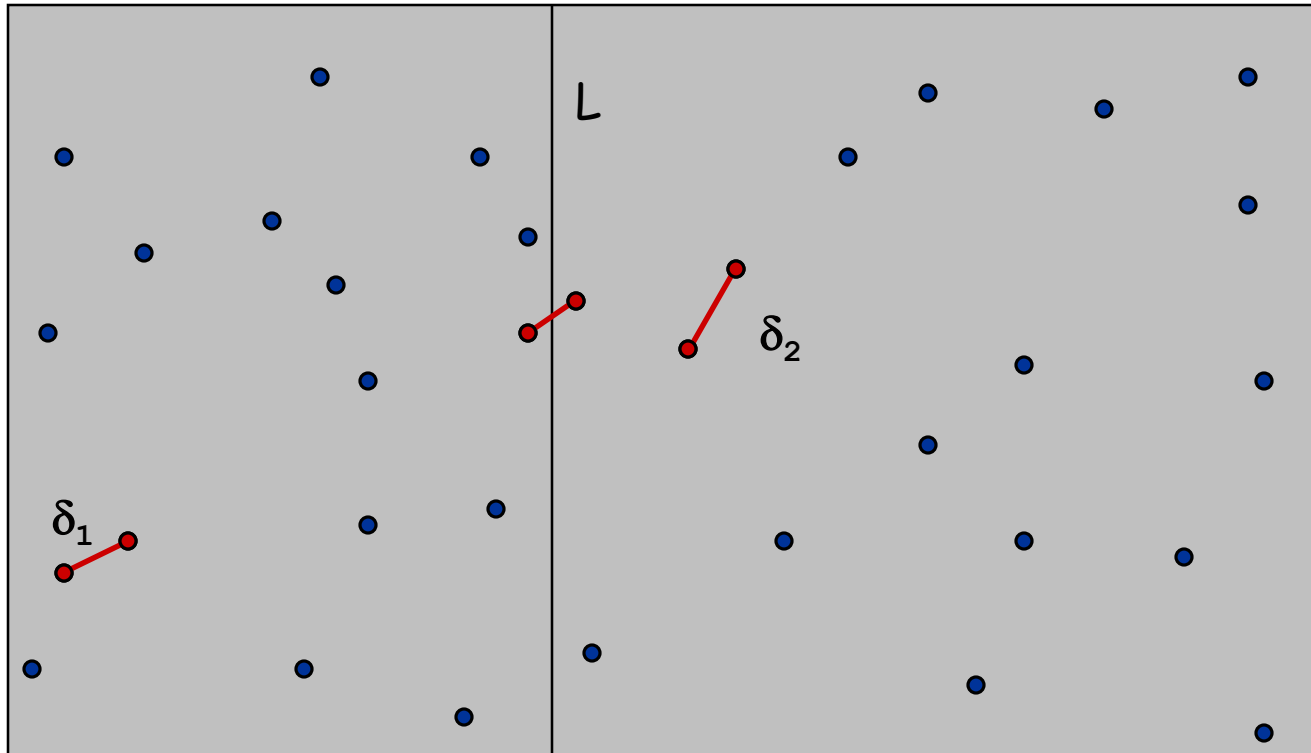
- Divide: draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.
- **Conquer**: find closest pair in each side recursively.



Closest Pair of Points

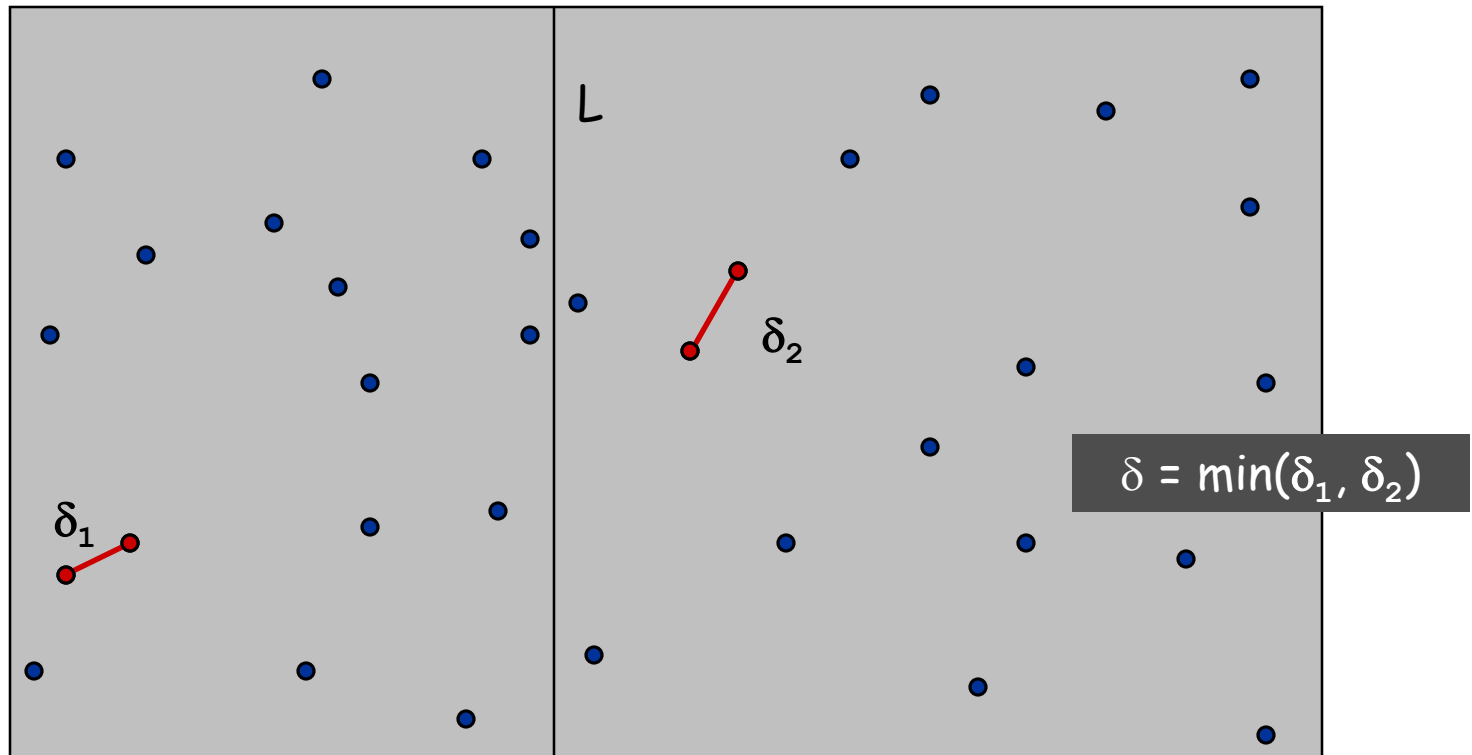
Algorithm.

- 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

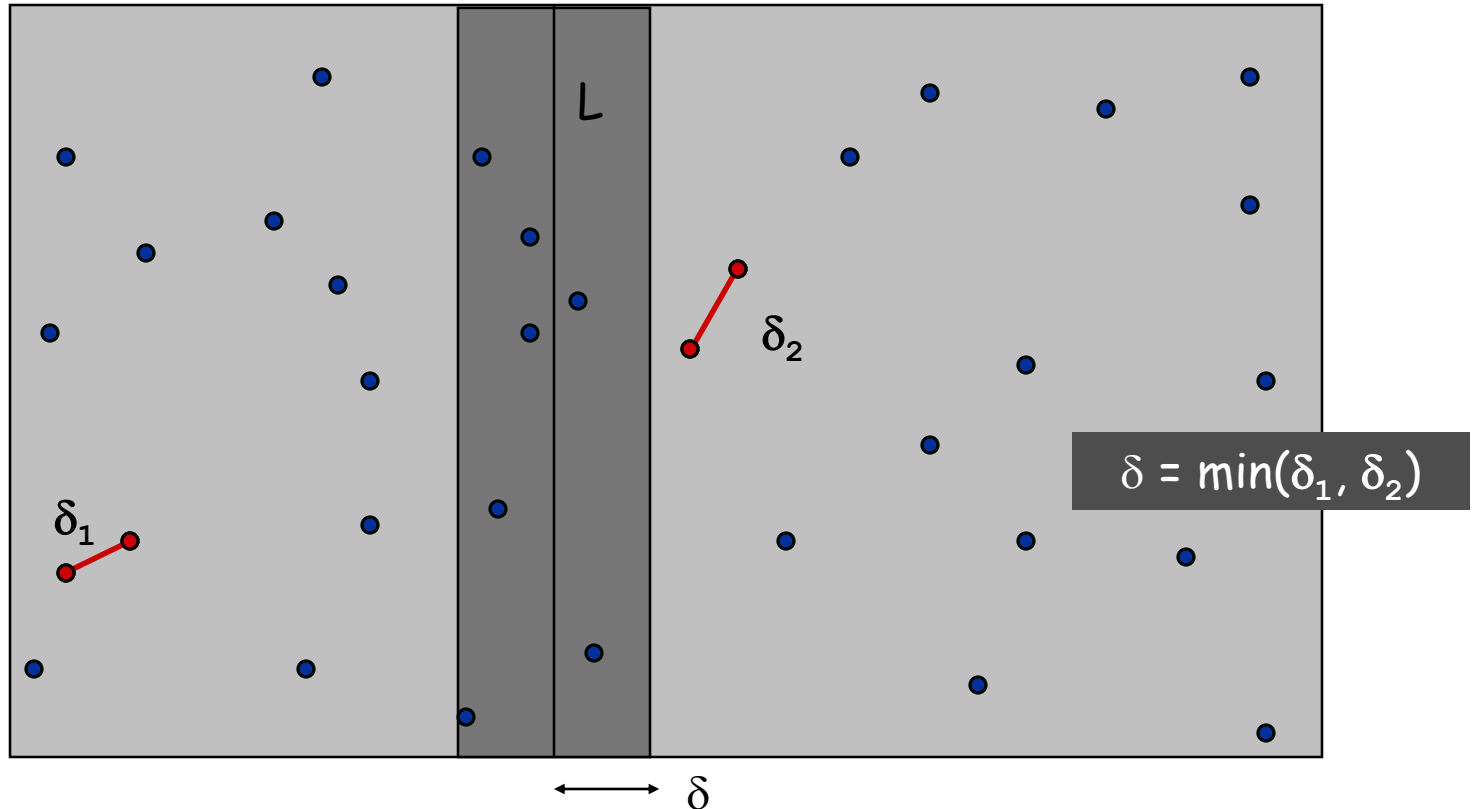
Find closest pair with one point in each side, **assuming that distance $< \delta$** .



Closest Pair of Points

Find closest pair with one point in each side, **assuming that distance $< \delta$** .

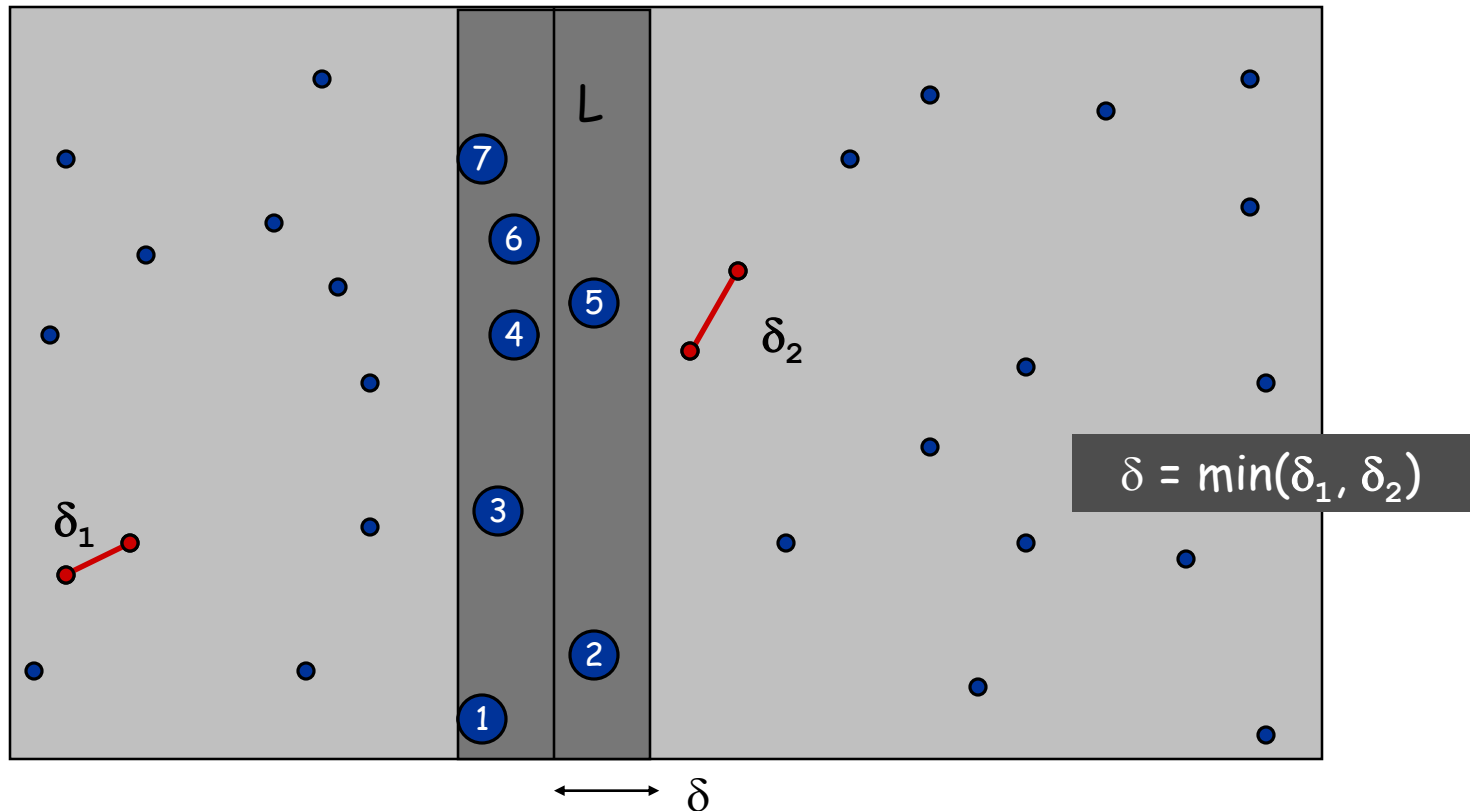
- Observation: only need to consider points within δ of line L .



Closest Pair of Points

Find closest pair with one point in each side, **assuming that distance $< \delta$** .

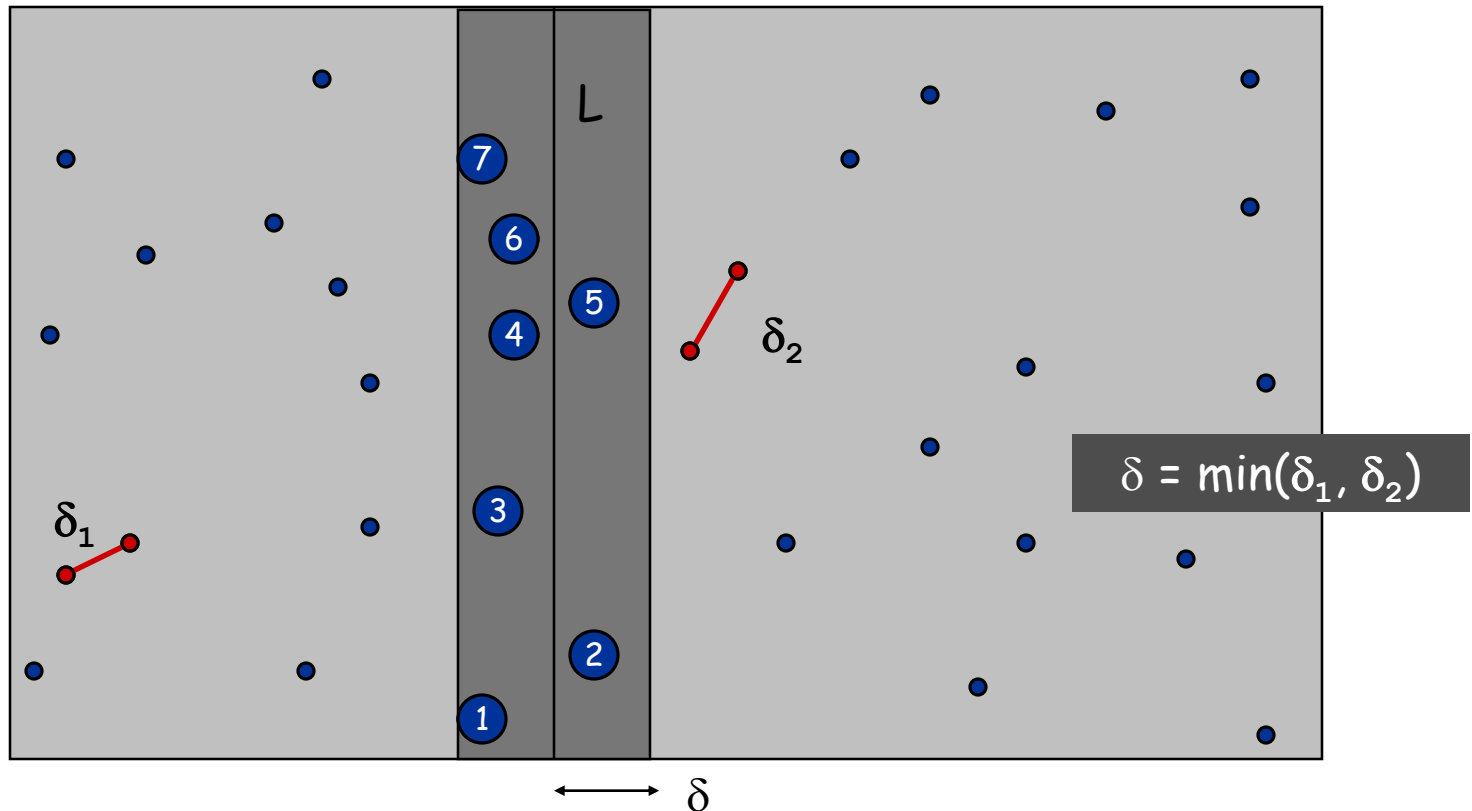
- Observation: only need to consider points within δ of line L .
- Sort points in 2δ -strip by their y coordinate.



Closest Pair of Points

Find closest pair with one point in each side, **assuming that distance $< \delta$** .

- 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

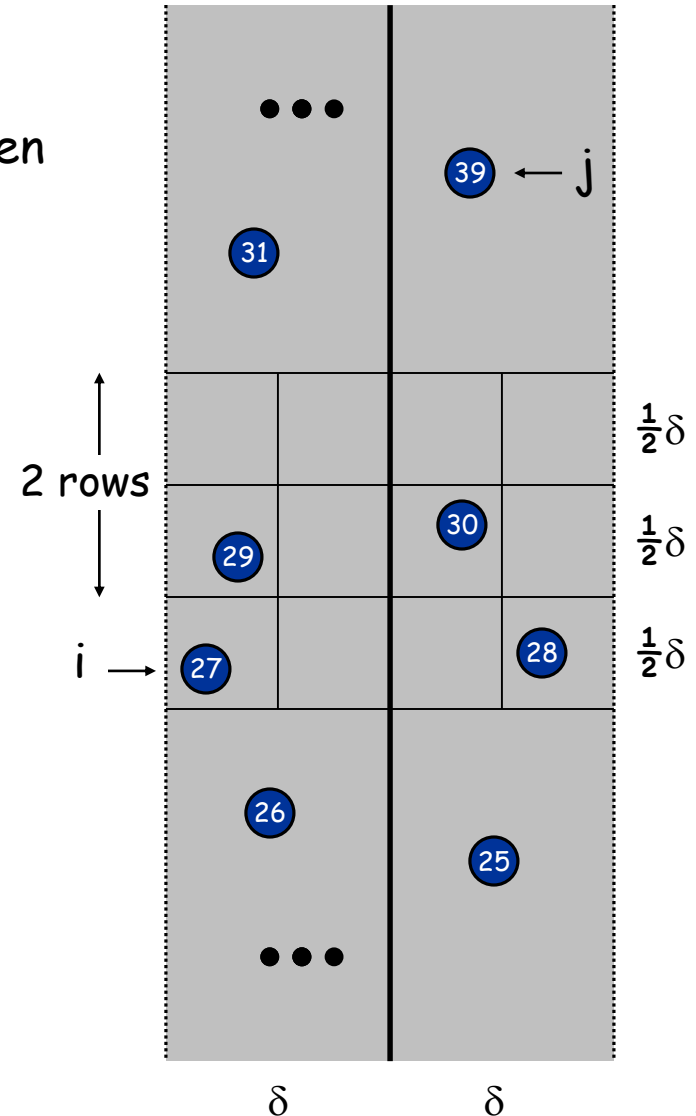
Def. Let s_i be the point in the 2δ -strip, with the i^{th} smallest y -coordinate.

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( $p_1, \dots, p_n$ ) {  
    Compute separation line  $L$  such that half the points  
    are on one side and half on the other side.  $O(n)$   
  
     $\delta_1 = \text{Closest-Pair}(\text{left half})$   
     $\delta_2 = \text{Closest-Pair}(\text{right half})$   $2T(n/2)$   
     $\delta = \min(\delta_1, \delta_2)$   
  
    Delete all points further than  $\delta$  from separation line  $L$   $O(n)$   
  
    Sort remaining points by  $y$ -coordinate.  $O(n \log n)$   
  
    Scan points in  $y$ -order and compare distance between  
    each point and next 11 neighbors. If any of these  
    distances is less than  $\delta$ , update  $\delta$ .  $O(n)$   
  
    return  $\delta$ .  
}
```

Closest Pair of Points: Analysis

Running time.

$$T(n) \leq 2T(n/2) + O(n \log n) \Rightarrow T(n) = O(n \log^2 n)$$

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.

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

Integer Multiplication

Integer Addition

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

Grade-school. $\Theta(n)$ bit operations.

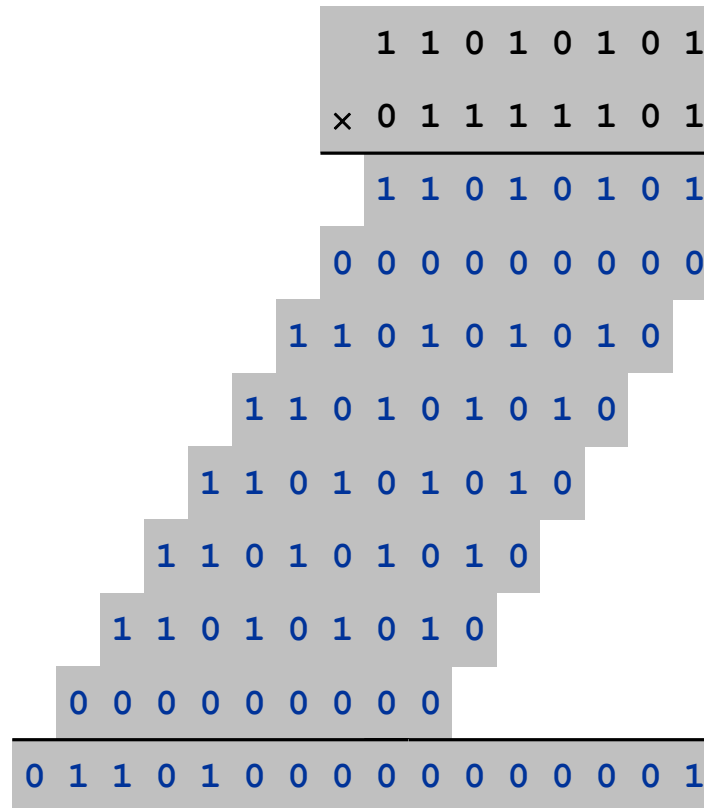
	1	1	1	1	1	1	0	1	
		1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	1	0	1
<hr/>									
	1	0	1	0	1	0	0	1	0

Remark. Grade-school addition algorithm is optimal.

Integer Multiplication

Multiplication. Given two n -bit integers a and b , compute $a \times b$.

Grade-school. $\Theta(n^2)$ bit operations.



Q. Is grade-school multiplication algorithm optimal?

Divide-and-Conquer Multiplication: Warmup

To multiply two n -bit integers a and b :

- Multiply four $\frac{1}{2}n$ -bit integers, recursively.
- Add and shift to obtain result.

$$\begin{aligned}a &= 2^{n/2} \cdot a_1 + a_0 \\b &= 2^{n/2} \cdot b_1 + b_0 \\ab &= \left(2^{n/2} \cdot a_1 + a_0\right)\left(2^{n/2} \cdot b_1 + b_0\right) = 2^n \cdot a_1 b_1 + 2^{n/2} \cdot (a_1 b_0 + a_0 b_1) + a_0 b_0\end{aligned}$$

Ex.

$$\begin{array}{ccc}a = & \underbrace{10001101}_{a_1} & \underbrace{1101101}_{a_0} \\b = & \underbrace{11100001}_{b_1} & \underbrace{1101101}_{b_0}\end{array}$$

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

Karatsuba Multiplication

To multiply two n -bit integers a and b :

- Add two $\frac{1}{2}n$ bit integers.
- Multiply **three** $\frac{1}{2}n$ -bit integers, recursively.
- Add, subtract, and shift to obtain result.

$$a = 2^{n/2} \cdot a_1 + a_0$$

$$b = 2^{n/2} \cdot b_1 + b_0$$

$$ab = 2^n \cdot a_1 b_1 + 2^{n/2} \cdot (a_1 b_0 + a_0 b_1) + a_0 b_0$$

$$= 2^n \cdot a_1 b_1 + 2^{n/2} \cdot ((a_1 + a_0)(b_1 + b_0) - a_1 b_1 - a_0 b_0) + a_0 b_0$$

1

2

1

3

3

Matrix Multiplication

Dot Product

Dot product. Given two length n vectors a and b , compute $c = a \cdot b$.

Grade-school. $\Theta(n)$ arithmetic operations.

$$a \cdot b = \sum_{i=1}^n a_i b_i$$

$$a = [.70 \quad .20 \quad .10]$$

$$b = [.30 \quad .40 \quad .30]$$

$$a \cdot b = (.70 \times .30) + (.20 \times .40) + (.10 \times .30) = .32$$

Remark. Grade-school dot product algorithm is optimal.

Matrix Multiplication

Matrix multiplication. Given two n -by- n matrices A and B , compute $C = AB$.

Grade-school. $\Theta(n^3)$ arithmetic operations.

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$\begin{bmatrix} .59 & .32 & .41 \\ .31 & .36 & .25 \\ .45 & .31 & .42 \end{bmatrix} = \begin{bmatrix} .70 & .20 & .10 \\ .30 & .60 & .10 \\ .50 & .10 & .40 \end{bmatrix} \times \begin{bmatrix} .80 & .30 & .50 \\ .10 & .40 & .10 \\ .10 & .30 & .40 \end{bmatrix}$$

Q. Is grade-school matrix multiplication algorithm optimal?

Block Matrix Multiplication

The diagram shows the following matrices and their partitions:

- C_{11} (red shaded): $\begin{bmatrix} 152 & 158 & 164 & 170 \\ 504 & 526 & 548 & 570 \\ 856 & 894 & 932 & 970 \\ 1208 & 1262 & 1316 & 1370 \end{bmatrix}$
- A_{11} (purple shaded): $\begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix}$
- A_{12} (purple shaded): $\begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix}$
- B_{11} (green shaded): $\begin{bmatrix} 16 & 17 \\ 20 & 21 \\ 24 & 25 \\ 28 & 29 \end{bmatrix}$
- B_{21} (green shaded): $\begin{bmatrix} 18 & 19 \\ 22 & 23 \\ 26 & 27 \\ 30 & 31 \end{bmatrix}$

The equation shown is: $C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix} = \begin{bmatrix} 152 & 158 \\ 504 & 526 \end{bmatrix}$$

Matrix Multiplication: Warmup

To multiply two n -by- n matrices A and B :

- Divide: partition A and B into $\frac{1}{2}n$ -by- $\frac{1}{2}n$ blocks.
- Conquer: multiply 8 pairs of $\frac{1}{2}n$ -by- $\frac{1}{2}n$ matrices, recursively.
- Combine: add appropriate products using 4 matrix additions.

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

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

$$T(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, form submatrices}} \Rightarrow T(n) = \Theta(n^3)$$

Strassen's Algorithm

Key idea. multiply 2-by-2 blocks with only **7 multiplications**.

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

$$C_{11} = P_5 + P_4 - P_2 + P_6$$

$$C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4$$

$$C_{22} = P_5 + P_1 - P_3 - P_7$$

$$P_1 = A_{11} \times (B_{12} - B_{22})$$

$$P_2 = (A_{11} + A_{12}) \times B_{22}$$

$$P_3 = (A_{21} + A_{22}) \times B_{11}$$

$$P_4 = A_{22} \times (B_{21} - B_{11})$$

$$P_5 = (A_{11} + A_{22}) \times (B_{11} + B_{22})$$

$$P_6 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$$

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

- 7 multiplications.
- $18 = 8 + 10$ additions and subtractions.

Strassen's algorithm works for any ring, such as plus/multiply, but not all semirings, such as min-plus or boolean algebra, where the naive algorithm still works.

Strassen's Algorithm

To multiply two n -by- n matrices A and B

- Divide: partition A and B into $\frac{1}{2}n$ -by- $\frac{1}{2}n$ blocks.
- Compute: 14 $\frac{1}{2}n$ -by- $\frac{1}{2}n$ matrices via 10 matrix additions.
- Conquer: multiply 7 pairs of $\frac{1}{2}n$ -by- $\frac{1}{2}n$ matrices, recursively.
- Combine: 7 products into 4 terms using 8 matrix additions.

Analysis.

- Assume n is a power of 2.
- $T(n) = \#$ arithmetic operations.

$$T(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \Rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

Fast Matrix Multiplication: Theory

Better results:

- $O(n^{2.376})$ [Coppersmith-Winograd, 1987]
- $O(n^{2.374})$ [Stothers, 2010]
- $O(n^{2.3728642})$ [Virginia Vassilevska Williams, 2011]
- $O(n^{2.3728639})$ [François Le Gall, 2014]

Conjecture. $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$.

Caveat. Theoretical improvements to Strassen are progressively less practical.

Maximum Sum Subarray

Maximum Sum Subarray

Problem: Given a one dimensional array $A[1..n]$ of numbers. Find a contiguous subarray with largest sum within A .

Assume an empty subarray has sum 0.

Example:



Algorithm (brute-force)

Observation: Let $S[i] = A[1] + \dots + A[i]$. We have $A[i] + \dots + A[j] = S[j] - S[i-1]$

```
Pre-Processing
S[0] = 0
for i = 1 to n do
    S[i] = S[i-1] + A[i]
```

Running time of pre-processing: $T(n) = O(n)$

```
sol = 0
for i = 1 to n do
    for j = i to n do
        if S[j] - S[i-1] > sol then
            sol = S[j] - S[i-1]
return sol
```

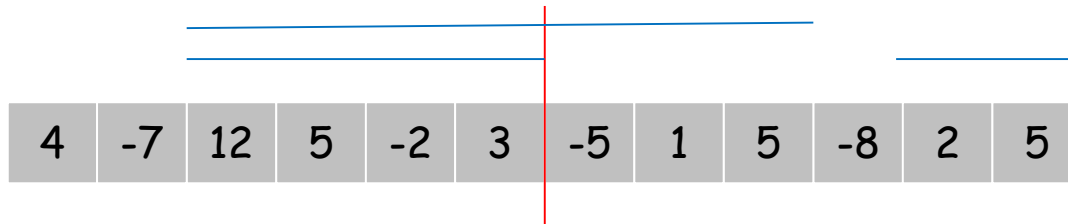
Running time: $T(n) = O(n^2)$

Algorithm (divide and conquer)

The general strategy: Divide into 2 equal-size subarrays

Case 1: optimal solution is in one subarray

Case 2: optimal solution crosses the splitting line



```
MCS(A[1..n])
if n = 1 then return max(0, a[1])
sol = max(MCS(A[1..n/2]), MCS(A[n/2+1..n]))
Lsol = 0
for i = n/2 downto 1 do
    if S[n/2]-S[i-1] > Lsol then
        Lsol = S[n/2]-S[i-1]
Rsol = 0
for i = n/2+1 to n do
    if S[i]-S[n/2-1] > solR then
        Rsol = S[i]-S[n/2-1]
return max(sol, Lsol+Rsol)
```

Running time: $T(n) = 2T\left(\frac{n}{2}\right) + O(n) \rightarrow T(n) = O(n \log n)$

References

References

- Sections 5.1, 5.2, 5.4, and 5.5 of the text book "algorithm design" by Jon Kleinberg and Eva Tardos
- Section 4.1 of the text book "introduction to algorithms" by CLRS, 3rd edition.
- The original slides were prepared by Kevin Wayne. The slides are distributed by Pearson Addison-Wesley.