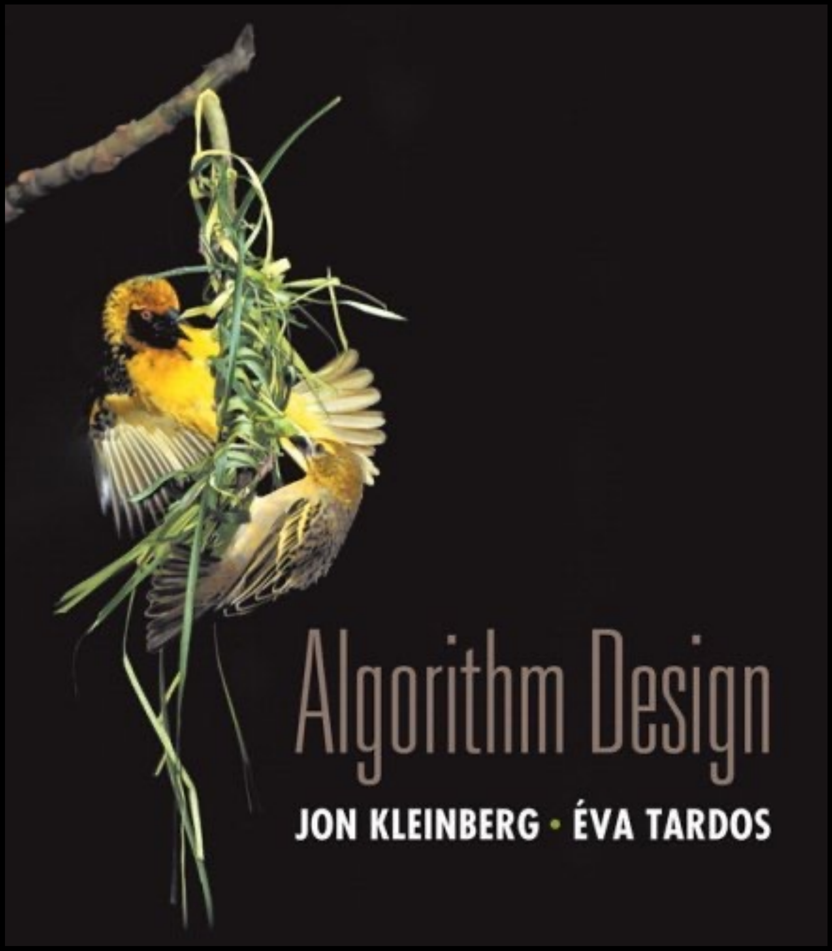


Chapter 5

Divide and Conquer

Slides by Kevin Wayne.

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.

5.1 Mergesort

Sorting

Sorting. Given n elements, rearrange in ascending order.

Applications.

- Sort a list of names.
- Organize an MP3 library.
- Display Google PageRank results.
- List RSS news items in reverse chronological order.

obvious applications

- Find the median.
- Find the closest pair.
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list.

problems become easy once items are in sorted order

- Data compression.
- Computer graphics.
- Computational biology.
- Supply chain management.
- Book recommendations on Amazon.
- Load balancing on a parallel computer.

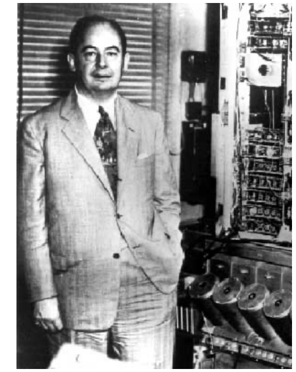
. . .

non-obvious applications

Mergesort

Mergesort.

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



Jon von Neumann (1945)

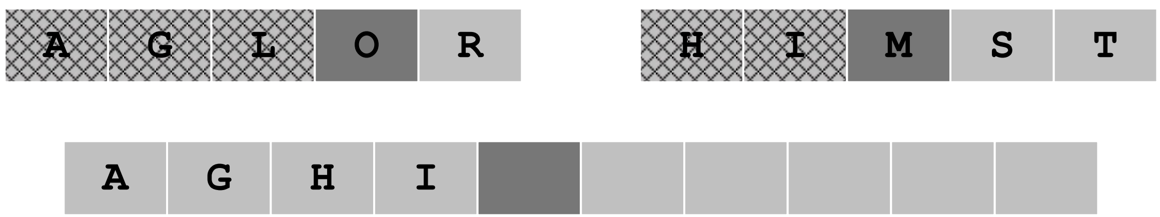
A L G O R I T H M S											
A L G O R	I T H M S	divide	$O(1)$								
A G L O R	H I M S T	sort	$2T(n/2)$								
A G H I L M O R S T		merge	$O(n)$								

Merging

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

How to merge efficiently?

- Linear number of comparisons.
- Use temporary array.



Challenge for the bored. In-place merge. [Kronrud, 1969]

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.

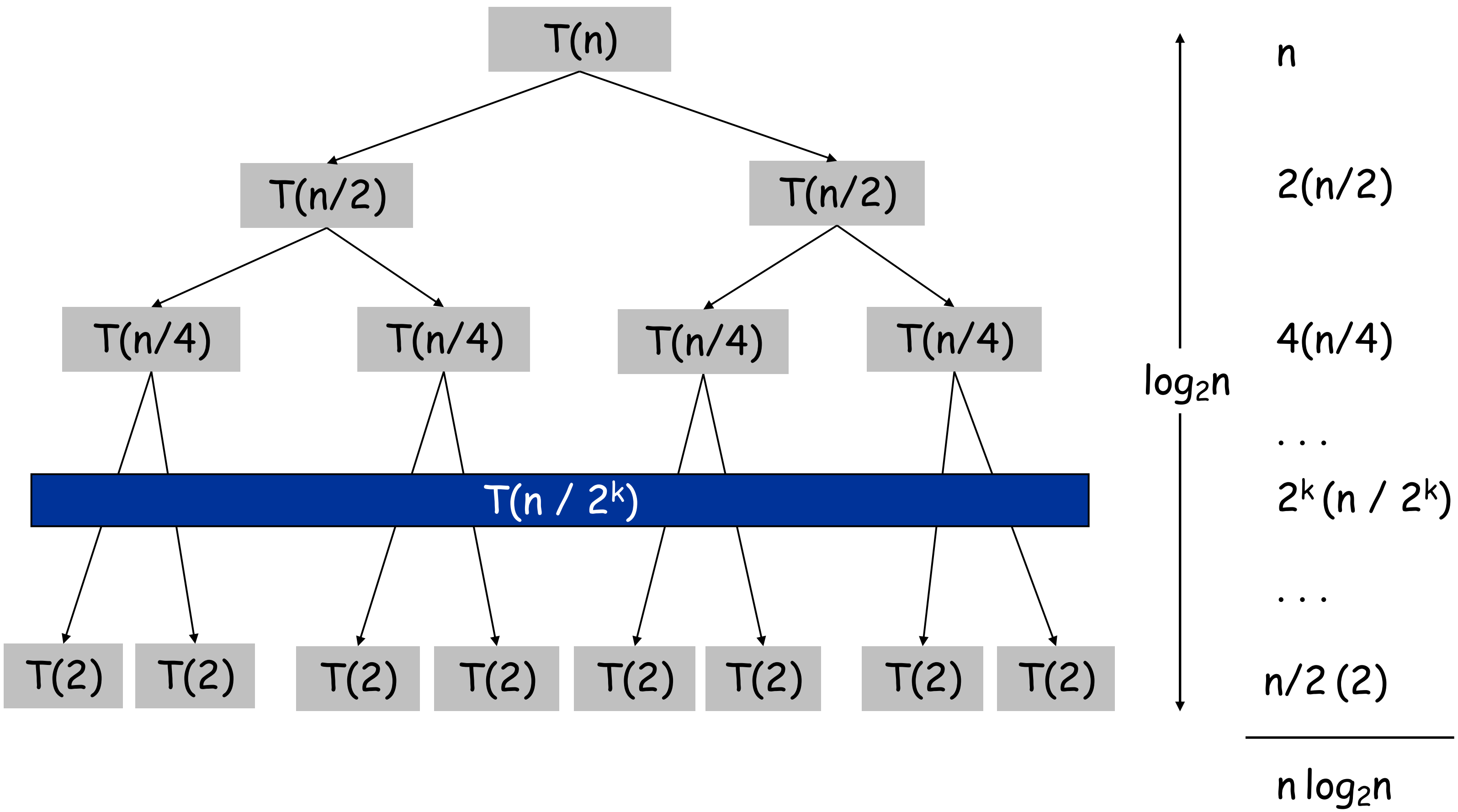
$$\mathrm{T}(n) \;\le\; \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T\left(\lceil n/2 \rceil\right)}_{\text{solve left half}} \;+\; \underbrace{T\left(\lfloor n/2 \rfloor\right)}_{\text{solve right half}} \;+\; \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

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

Assorted proofs. We describe several ways to prove this recurrence. Initially we assume n is a power of 2 and replace $\le$ with =.

Proof by Recursion Tree

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



Proof by Telescoping

Claim. If T(n) satisfies this recurrence, then T(n) = n $\log_2$ n.

assumes n is a power of 2

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

Pf. For n > 1:

$$\begin{aligned} \frac{T(n)}{n} &= \frac{2T(n/2)}{n} + 1 \\ &= \frac{T(n/2)}{n/2} + 1 \\ &= \frac{T(n/4)}{n/4} + 1 + 1 \\ &\cdots \\ &= \frac{T(n/n)}{n/n} + \underbrace{1 + \cdots + 1}_{\log_2 n} \\ &= \log_2 n \end{aligned}$$

Proof by Induction

Claim. If T(n) satisfies this recurrence, then $T(n) = n \log_2 n$.

assumes n is a power of 2

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

Pf. (by induction on n)

- Base case: $n = 1$.
- Inductive hypothesis: $T(n) = n \log_2 n$.
- Goal: show that $T(2n) = 2n \log_2 (2n)$.

$$\begin{aligned} T(2n) &= 2T(n) + 2n \\ &= 2n \log_2 n + 2n \\ &= 2n\left(\log_2 (2n) - 1\right) + 2n \\ &= 2n \log_2 (2n) \end{aligned}$$

Analysis of Mergesort Recurrence

Claim. If T(n) satisfies the following recurrence, then $T(n) \le n \lceil \lg n \rceil$.

$\lg n$: $\log_2 n$

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

Pf. (by induction on n)

- Base case: n = 1.
- Define $n_1 = \lfloor n / 2 \rfloor$, $n_2 = \lceil n / 2 \rceil$.
- Induction step: assume true for 1, 2, ... , n–1.

$$\begin{aligned} T(n) &\le T(n_1) + T(n_2) + n \\ &\le n_1 \lceil \lg n_1 \rceil + n_2 \lceil \lg n_2 \rceil + n \\ &\le n_1 \lceil \lg n_2 \rceil + n_2 \lceil \lg n_2 \rceil + n \\ &= n \lceil \lg n_2 \rceil + n \\ &\le n(\lceil \lg n \rceil - 1) + n \\ &= n \lceil \lg n \rceil \end{aligned}$$

$$\begin{aligned} n_2 &= \lceil n/2 \rceil \\ &\le \left\lceil 2^{\lceil \lg n \rceil} / 2 \right\rceil \\ &= 2^{\lceil \lg n \rceil} / 2 \\ \Rightarrow \lg n_2 &\le \lceil \lg n \rceil - 1 \end{aligned}$$

Two Exercises

- Using recursion tree to guess a result, and then, applying induction to prove.

(1) $T(n) = 3\,T(\lfloor n/4 \rfloor) + \Theta(n^2)$

Use cn^2 to replace $\Theta(n^2)$ for $c > 0$ in recursion tree

Apply $T(n) \le dn^2$ for $d > 0$, the guess result, in induction prove

Determine the constraint associated with d and c

(2) $T(n) = T(n/3) + T(2n/3) + O(n)$

Use c to represent the constant factor in $O(n)$ in recursion tree

Apply $T(n) \le d\, n \lg n$ for $d > 0$, the guess result, in induction prove

Determine the constraint associated with d and c

Master Theorem

The master theorem

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n),$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

$T(n) = 9\,T(n/3) + n$, $T(n) = \Theta(n^2)$;

$T(n) = T(2n/3) + 1$, $T(n) = \Theta(\log n)$;

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

$T(n) = 3T(n/4) + n \log n$, $T(n) = \Theta(n \log n)$

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

$T(n) = 7T(n/2) + \Theta(n^2)$, $T(n) = \Theta(n^{\log 7})$

Parallel Merge Sort

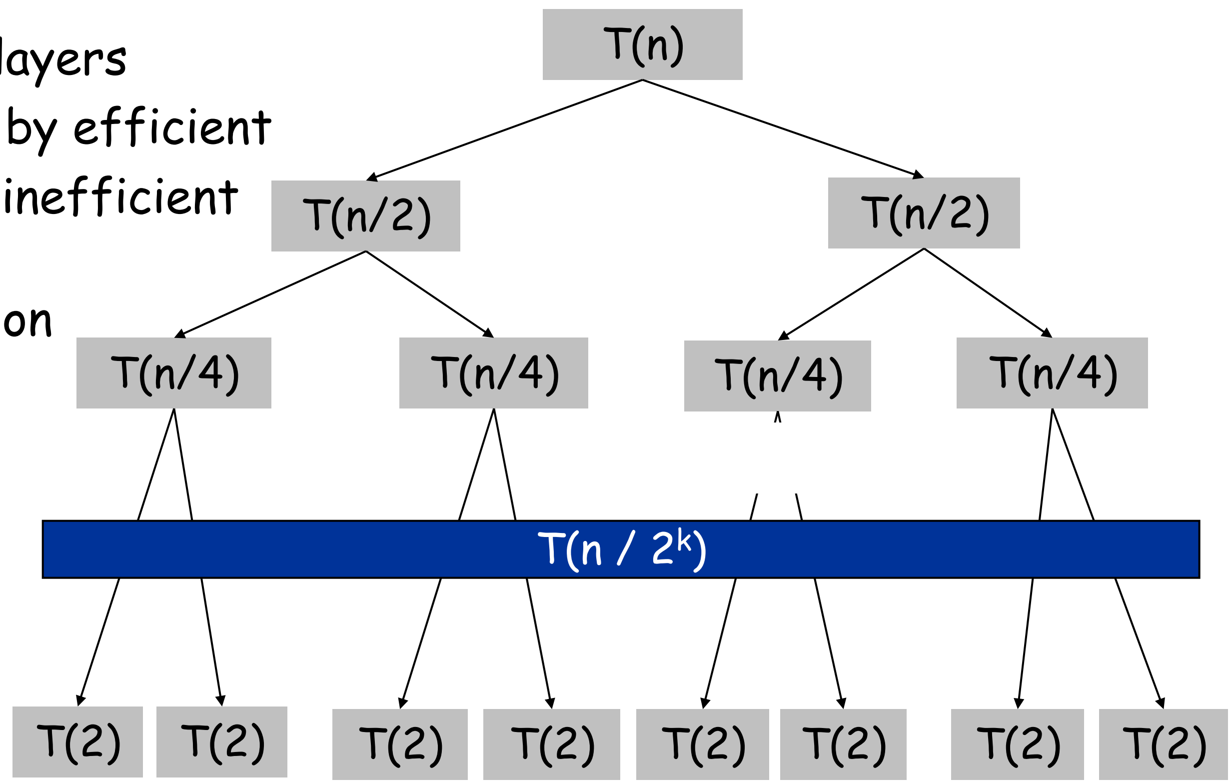
Merge sort with parallel recursion: $O(n)$, still slow

Parallel multiway merge sort

Merge sort with parallel merge

Merge sort with two layers
- Bottom layer: slow by efficient
- Top layer: fast but inefficient

Multisequence selection

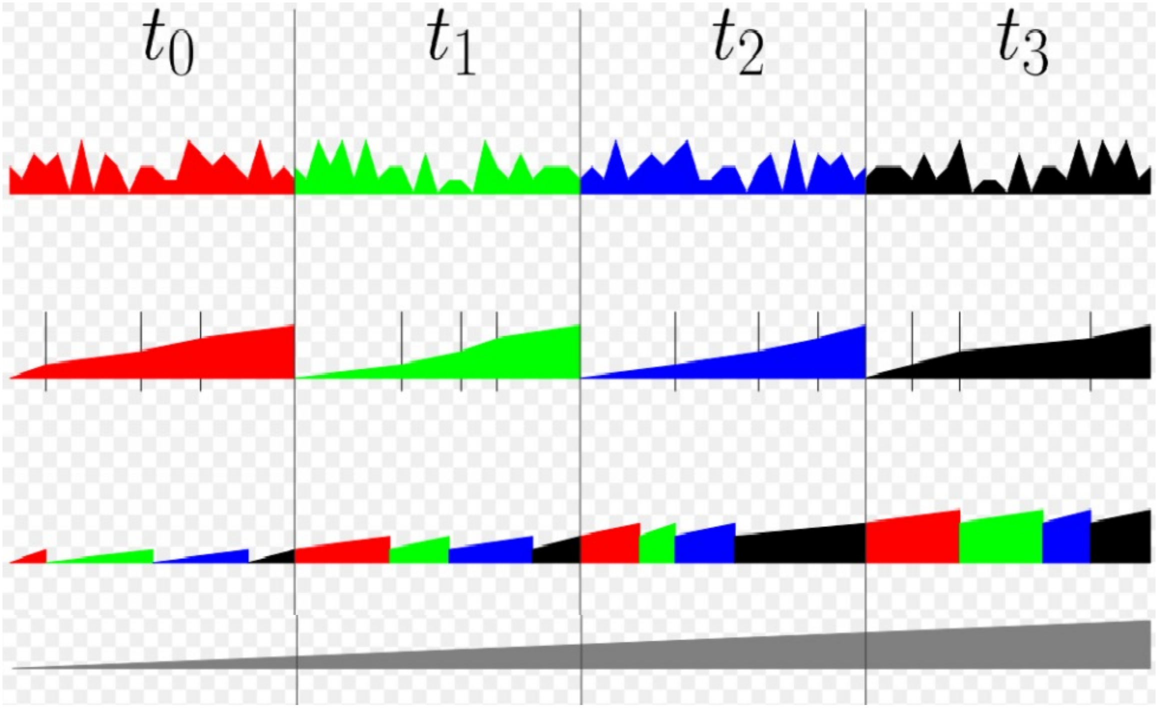


Extra: Parallel Multiway Merge Sort

Map-Shuffle-Reduce in Hadoop

- Partition data and assign to m processors
- Each processor sorts data based on n samples

Data access: message passing



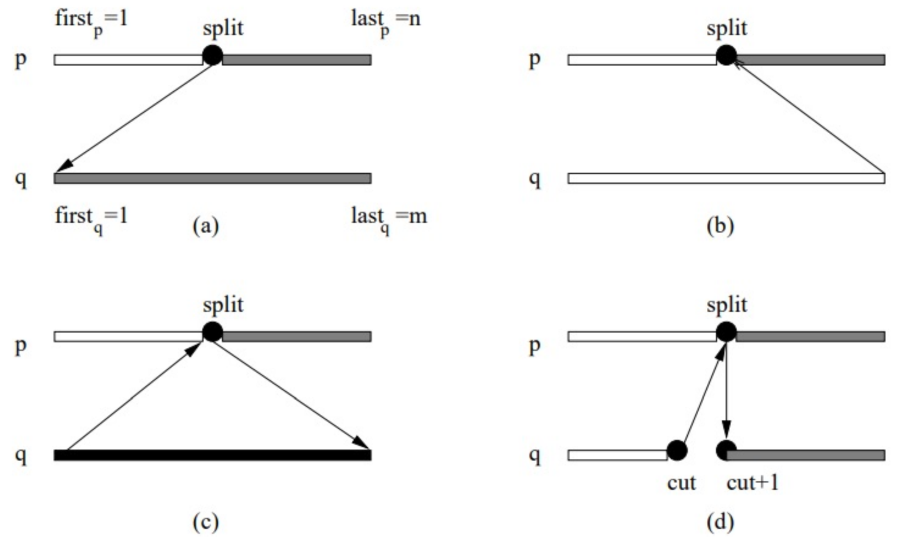
Extra: Parallel Merge (Sort)

Merge two sorted subsequences: $O(\log^2 n)$ with $O(n / \log^2 n)$ processors: switch to sequence merge sort with sizes are reduced to $O(\log^2 n)$

Data access: PRAM

Parallel random-access memory

EREW or CRCW



Speedup: seq. time / para. time, Efficiency: # of processors (k)/speed up

Cost: # of processors x parallel time, Cost-optimal: efficiency = 1

Other parallel sorts: bitonic, quick, radix, and sample sort

R. Cole, Parallel Merge Sort, SIAM Journal on Computing, 1988

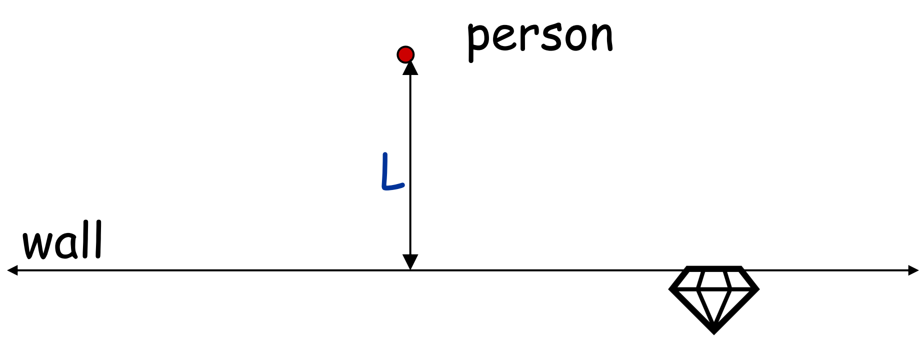
J. Wu and S. Olariu, On Cost-Optimal Merge of Two Intransitive Sorted Sequence, 2003

Extra: Searching

Systematically search the "space" for a solution.

Key: how to divide (-and-eliminate) the solution space.

1. A person is L distance away from a long wall with no end on both sides. A diamond is placed on the wall which can be identified through touching. Design a searching method with a constant bound in moving distance.



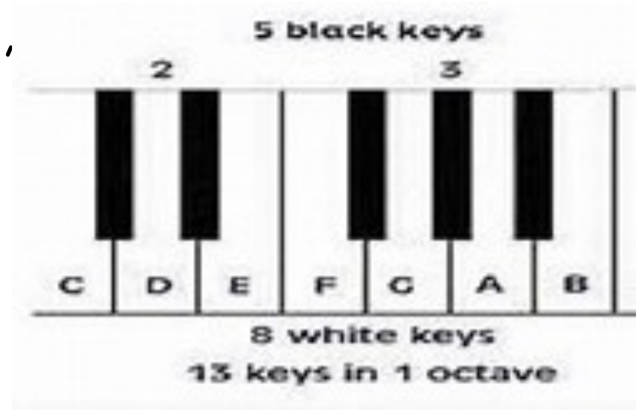
Extra: Searching

1. A fish needs to be steamed between 5 to 18 minutes. Design a fast-searching method to find the best cooking time. Under- and over-cook can be compared via tasting, but not during cooking. (1 minute is the basic unit of time duration. Quality of fish is a quadratic function.)

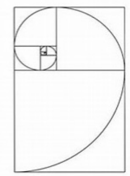
   check golden-section search

Extra: Fibonacci Sequence and Golden Ratio

$F_n=F_{n-1}+F_{n-2}$, $F_0=0$, $F_1=1$: 0, 1, 1, 2, 3, 5, 8, 13, 21,

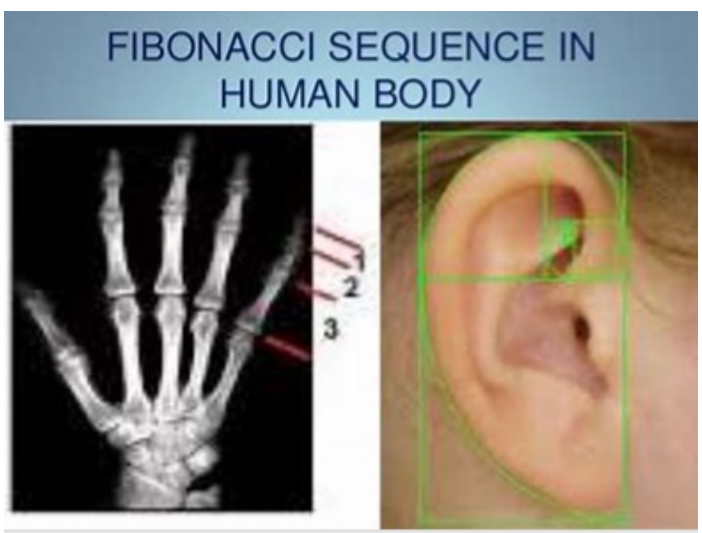


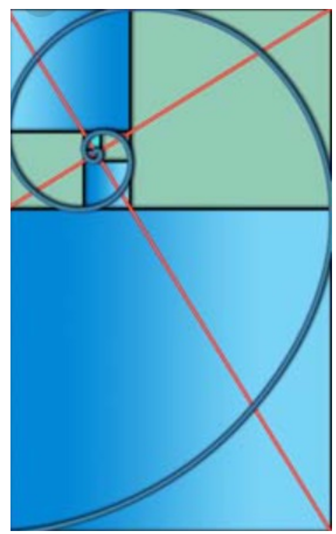
$(a+b)/b = b/a = 1.618…$



In music, human body, nature, …

Eye of god





Extra: Fibonacci Puzzle

Extended Fibonacci sequence:

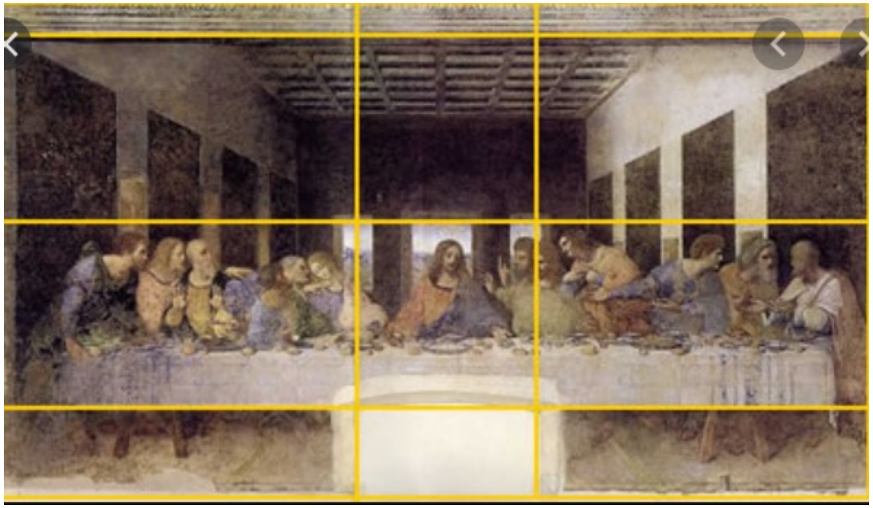
2, 4, 6, 10, 16, 26, …

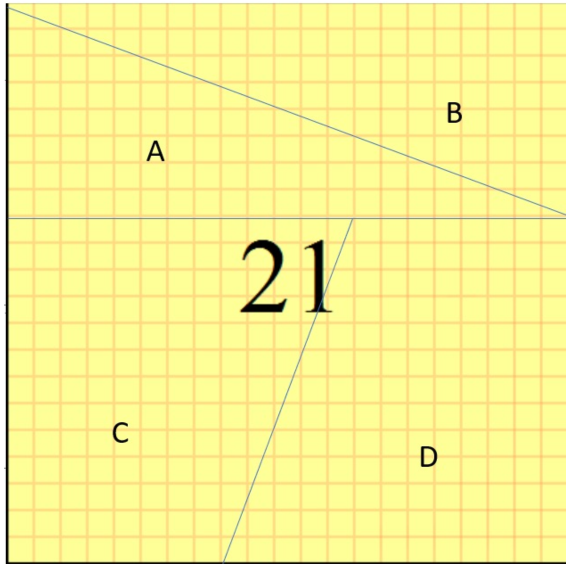
4, 8, 12, 20, 32, 52, …

8, 16, 24, 40, 64, 104, …

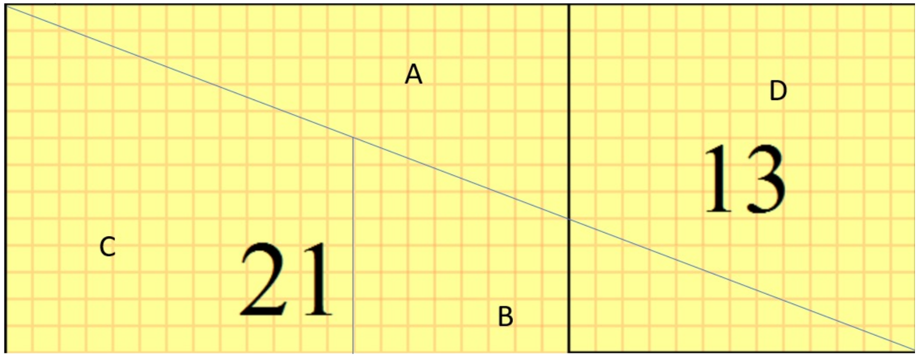
Fibonacci sequence in Last Super

1, 2, 3, 5, 8, 13





$21 \times 21 = 34 \times 13$ (?)



Mathematics is the language in which God has written the universe -Galileo Galilei

5.3 Counting Inversions

Counting Inversions

Music site tries to match your song preferences with others.

- You rank n songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.

- My rank: 1, 2, …, n.
- Your rank: $a_1, a_2, \ldots, a_n$.
- Songs i and j 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 i and j.

More Applications of Rankings

Applications.

- Collaborative filtering (preferences /taste info. from many users)
- Measuring the "sortedness" of an array.
- Sensitivity analysis of Google's ranking function.

Voting theory: 3-party voting (Condorcet paradox, Arrow's impossibility theorem on voting)

1: A>B>T Based on 1 and 3: A beats B

2: B>T>A Based on 2 and 3: T beats A

3: T>A>B Based on 1 and 2: B beats T

(A: Anderson, B: Biden, T: Trump)

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

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

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.

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

Divide: O(1).

1	5	4	8	10	2

6	9	12	11	3	7

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half.

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

Divide: O(1).

1	5	4	8	10	2

6	9	12	11	3	7

Conquer: 2T(n / 2)

5 blue-blue inversions

8 green-green inversions

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

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

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(1).

1	5	4	8	10	2

6	9	12	11	3	7

5 blue-blue inversions

8 green-green inversions

Conquer: 2T(n / 2)

9 blue-green inversions
5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

Combine: ???

Total = 5 + 8 + 9 = 22.

Counting Inversions: Combine

Combine: count blue-green inversions

- Assume each half is sorted.
- Count inversions where a_i and a_j are in different halves.
- Merge two sorted halves into sorted whole.

to maintain sorted invariant

3	7	10	14	18	19

2	11	16	17	23	25
6	3	2	2	0	0

13 blue-green inversions: 6 + 3 + 2 + 2 + 0 + 0

Count: O(n)

2	3	7	10	11	14	16	17	18	19	23	25

Merge: O(n)

$$T(n) \le T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n) \quad \Rightarrow \quad \mathrm{T}(n) = O(n \log n)$$

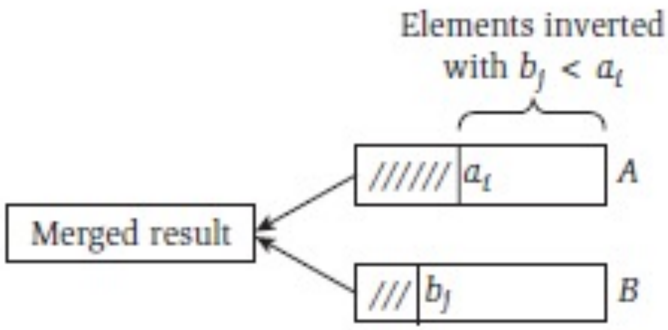
Counting Inversions: Implementation

Pre-condition. [Merge-and-Count] A and B are sorted.
Post-condition. [Sort-and-Count] L is sorted.

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

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

   return r = rA + rB + r and the sorted list L
}
```



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.

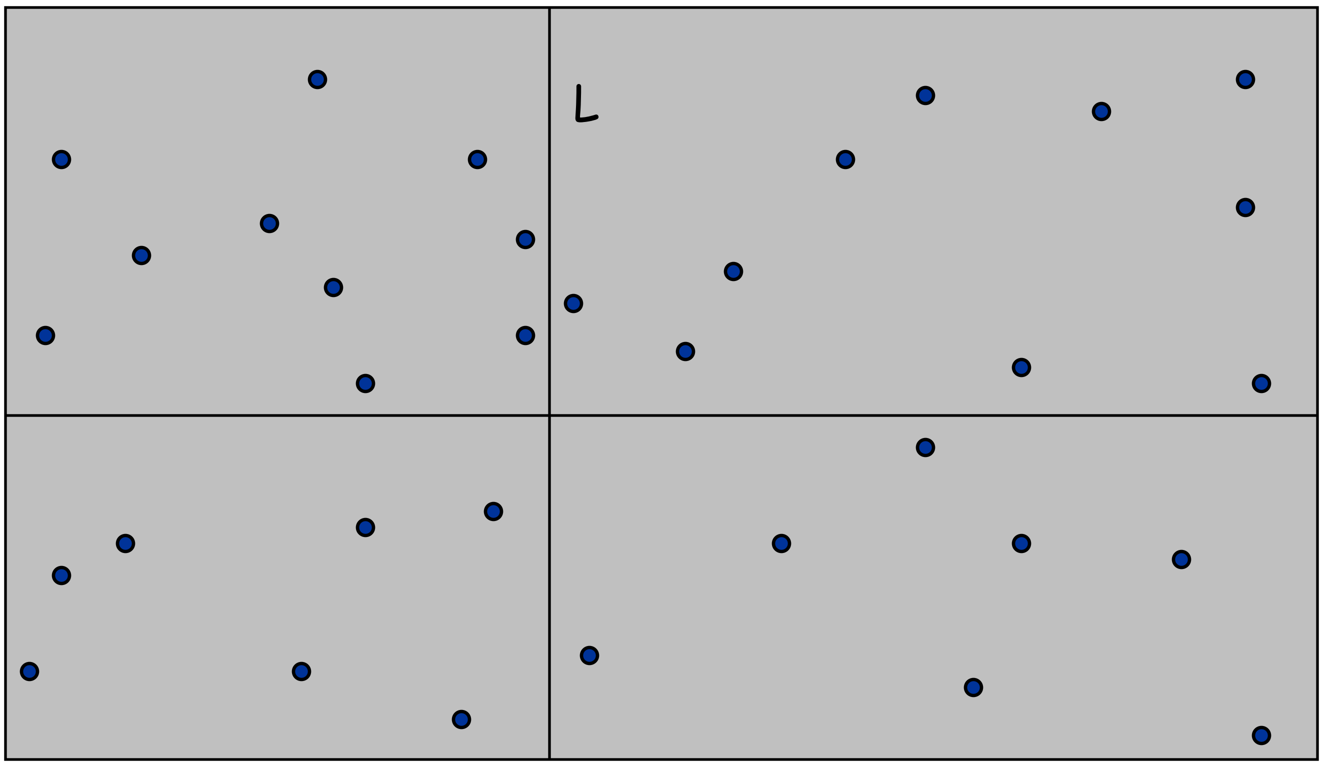
1-D version. O(n log n) easy if points are on a line.

Assumption. No two points have same x coordinate.

to make presentation cleaner

Closest Pair of Points: First Attempt

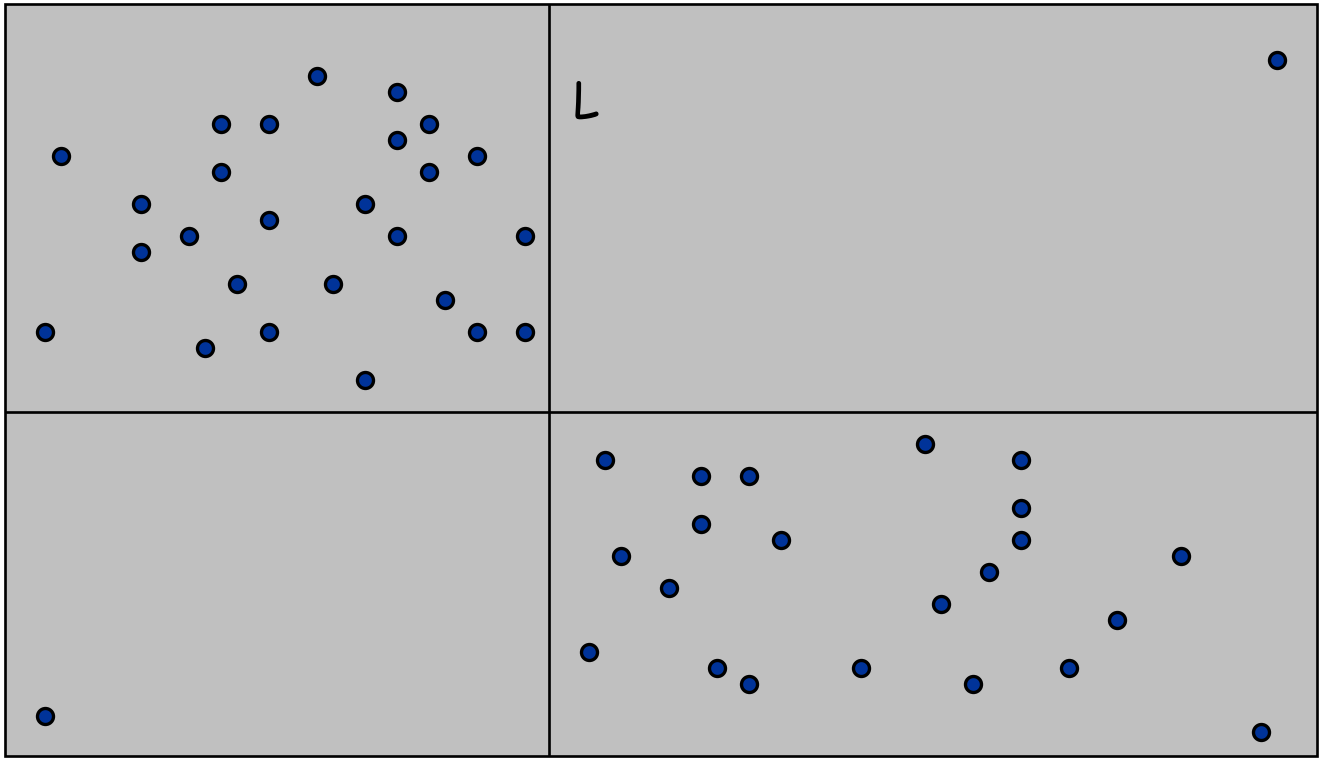
Divide. Sub-divide region into 4 quadrants.



Closest Pair of Points: First Attempt

Divide. Sub-divide region into 4 quadrants.

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

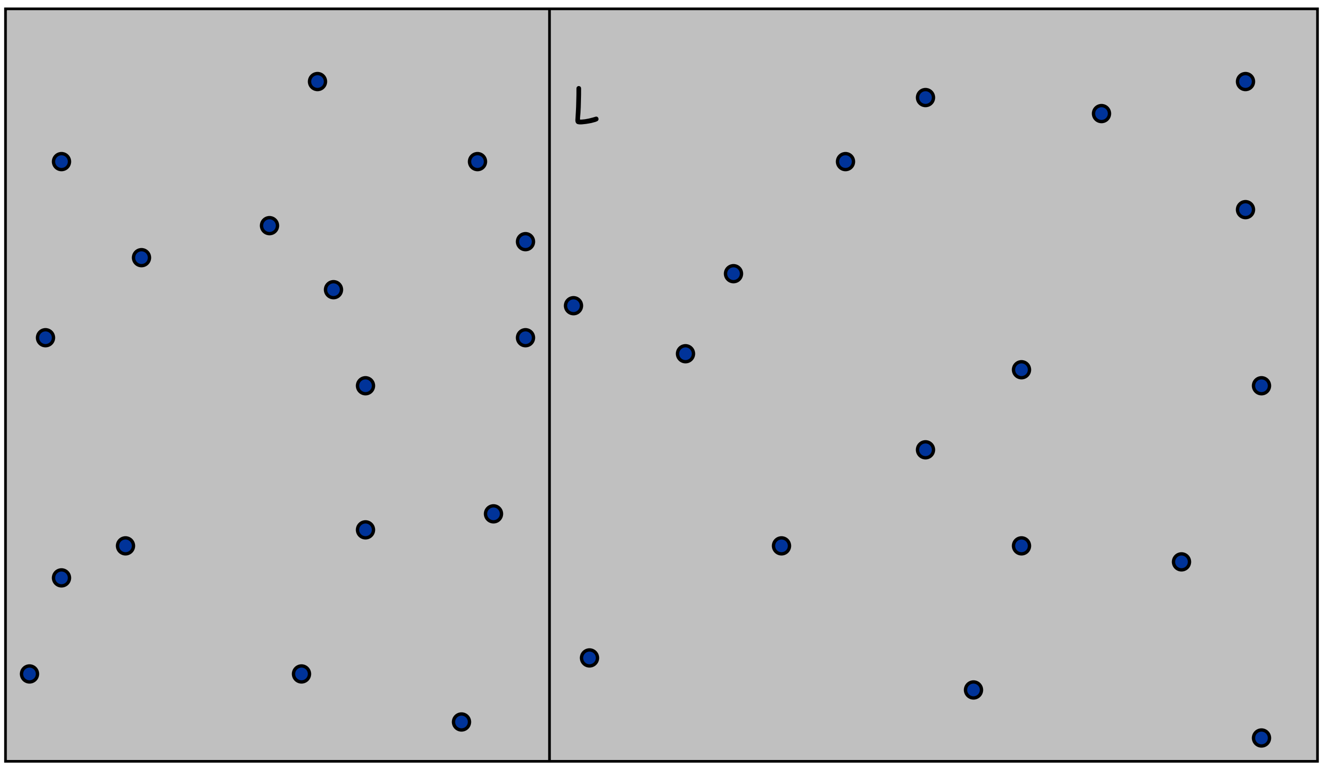


Closest Pair of Points

Algorithm.

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

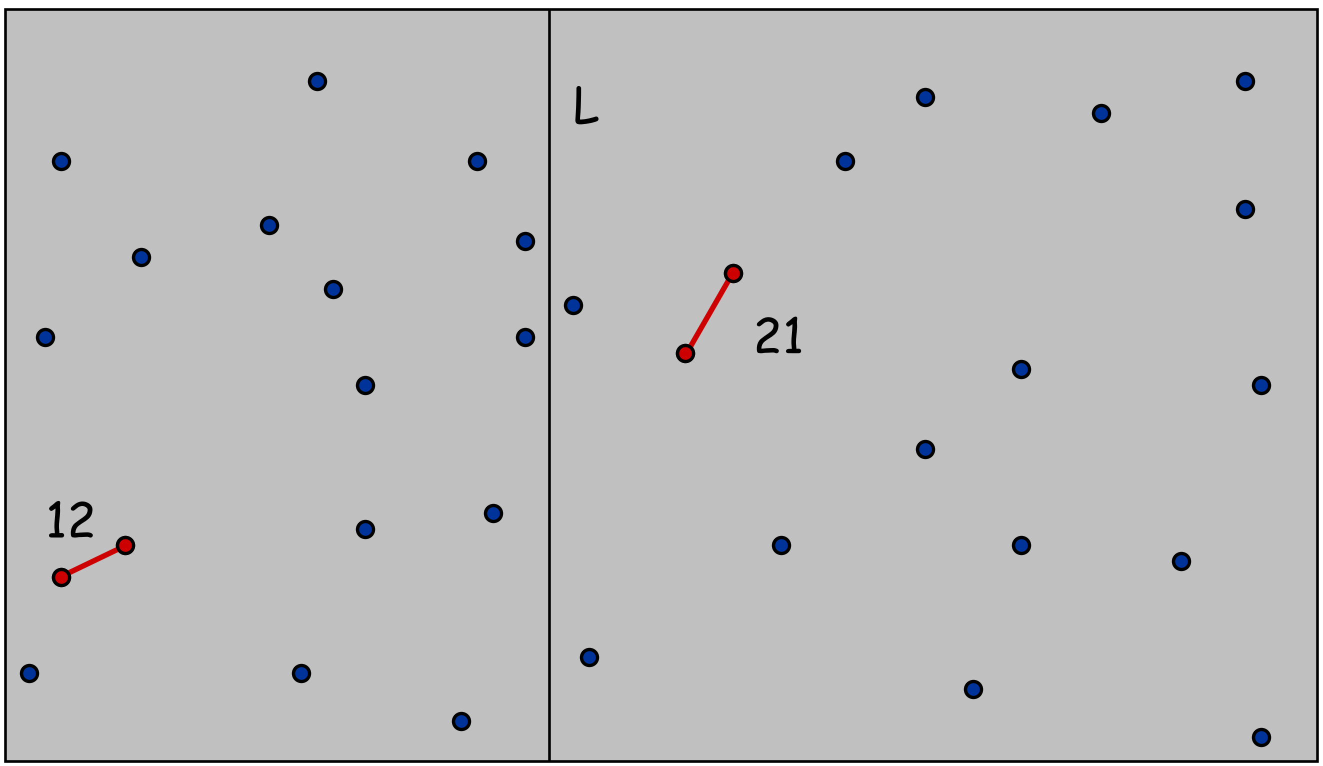
- Bonus point: Given a country map, perform two vertical-oriented cuts of the map into three parts: east, middle, and west, such that
  |east| + |west| = |middle| (|x| stands for the population of x)



Closest Pair of Points

Algorithm.

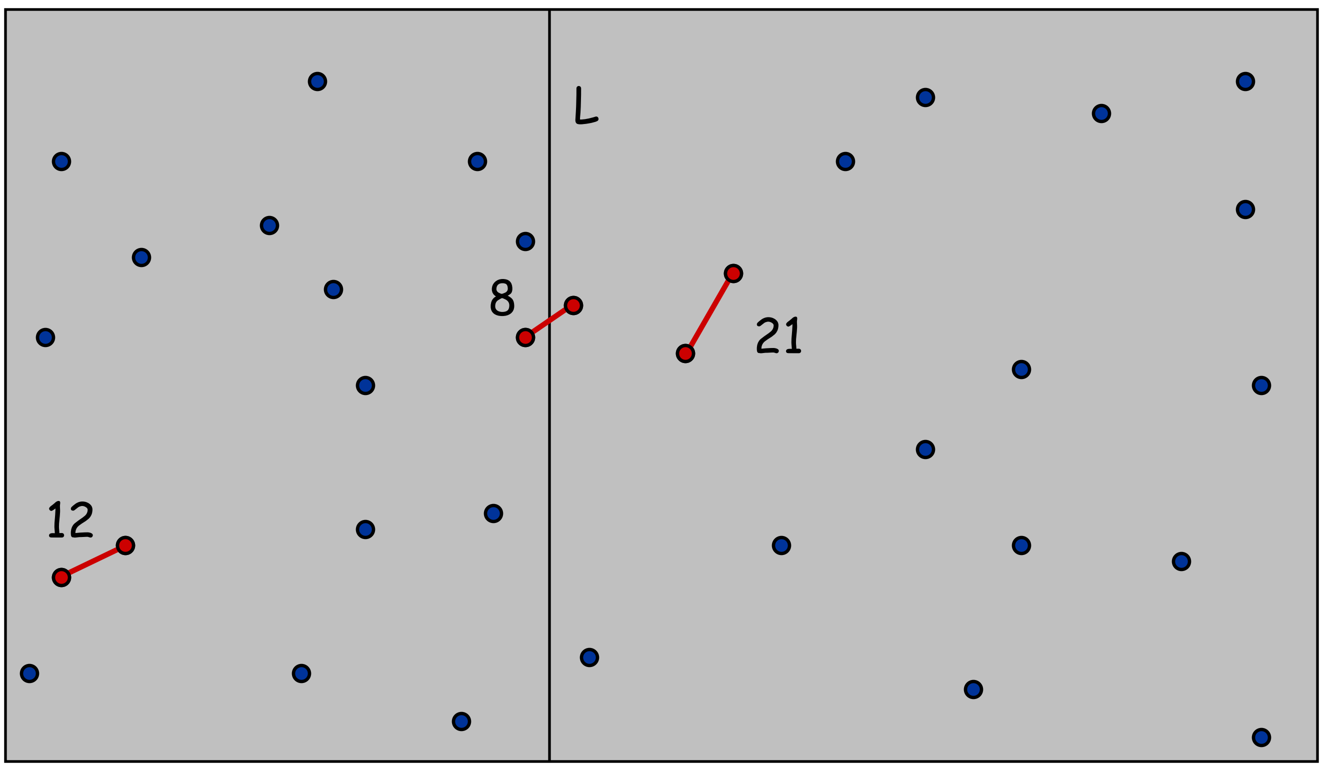
- Divide: draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.
- Conquer: find closest pair in each side recursively.



Closest Pair of Points

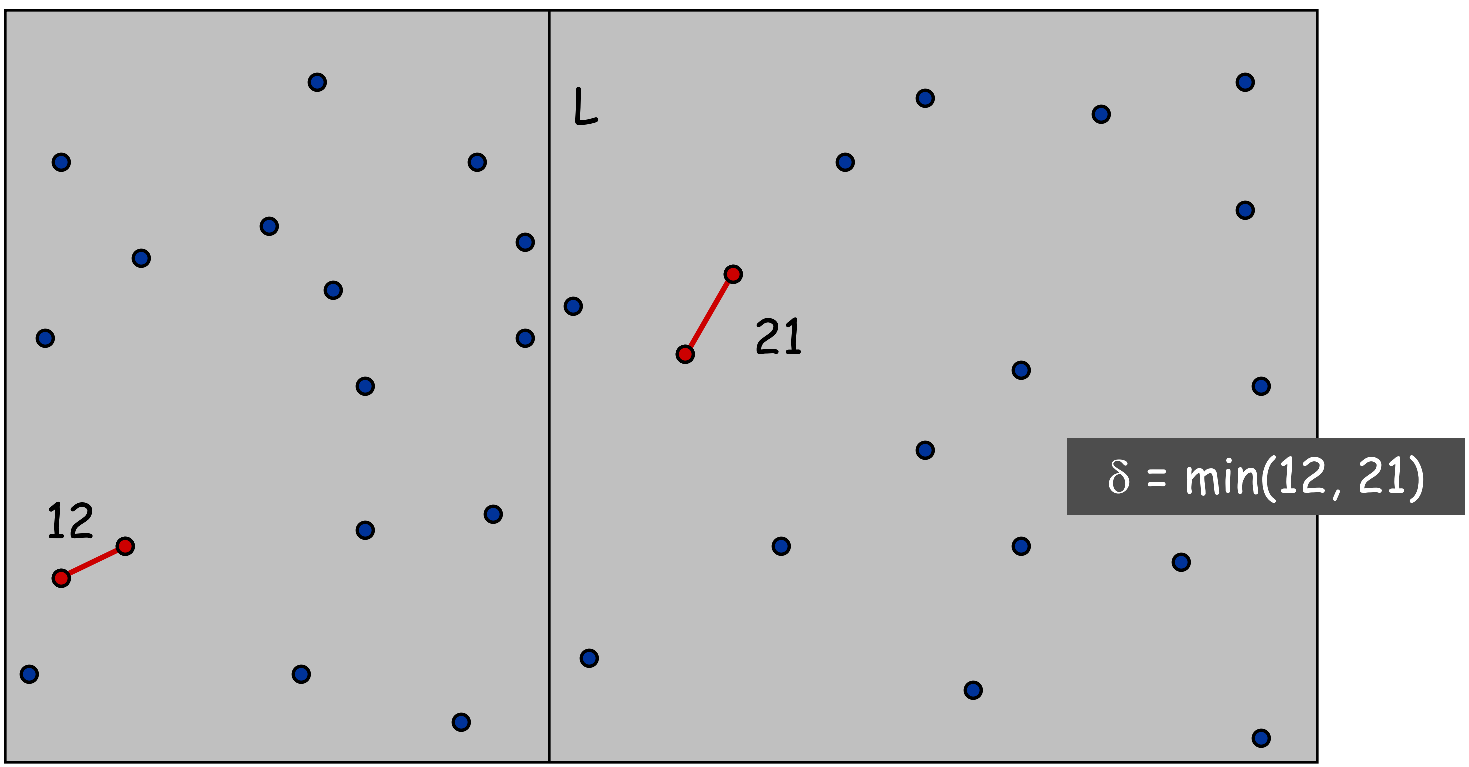
Algorithm.

- 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

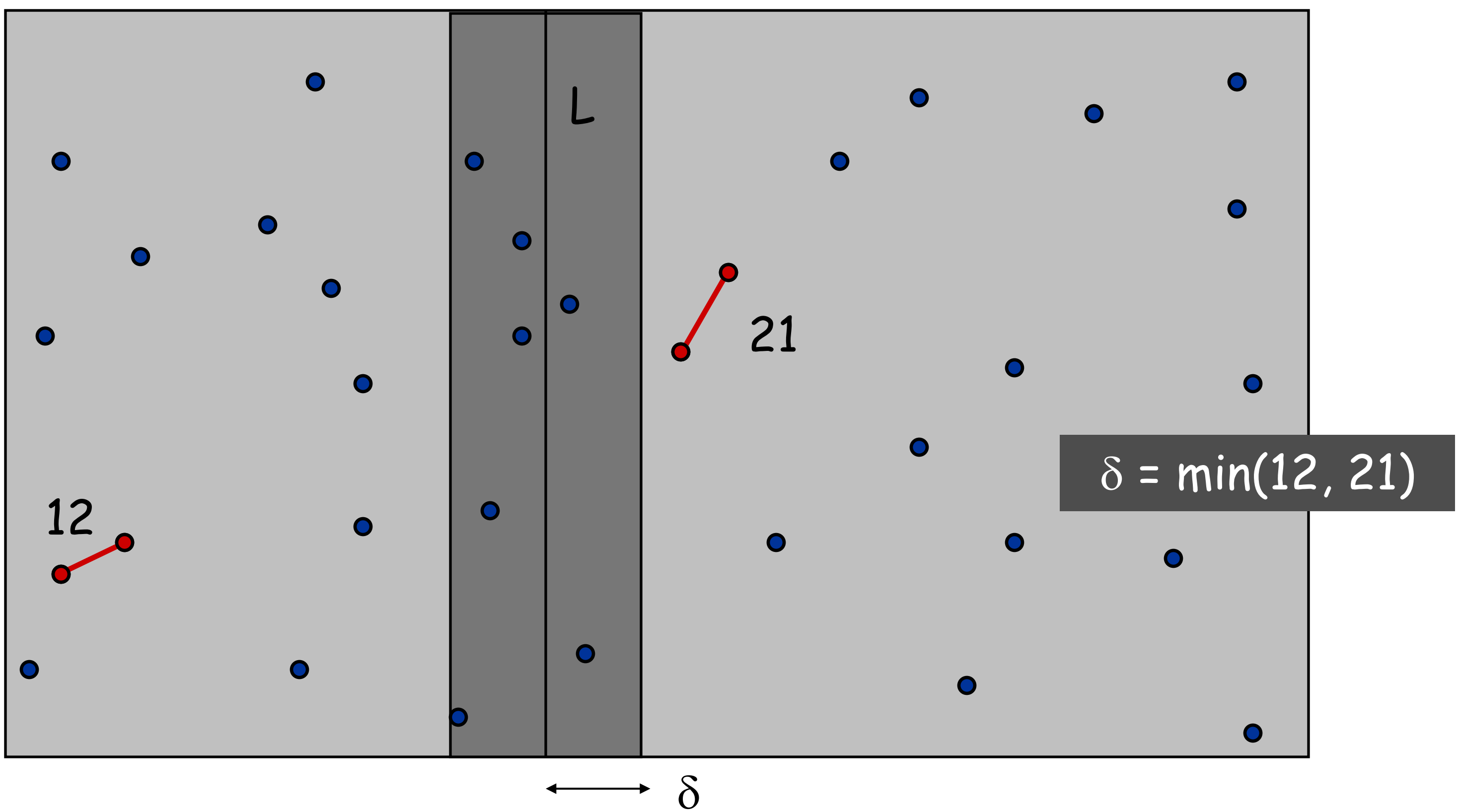
Find closest pair with one point in each side, assuming that distance < δ.



Closest Pair of Points

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

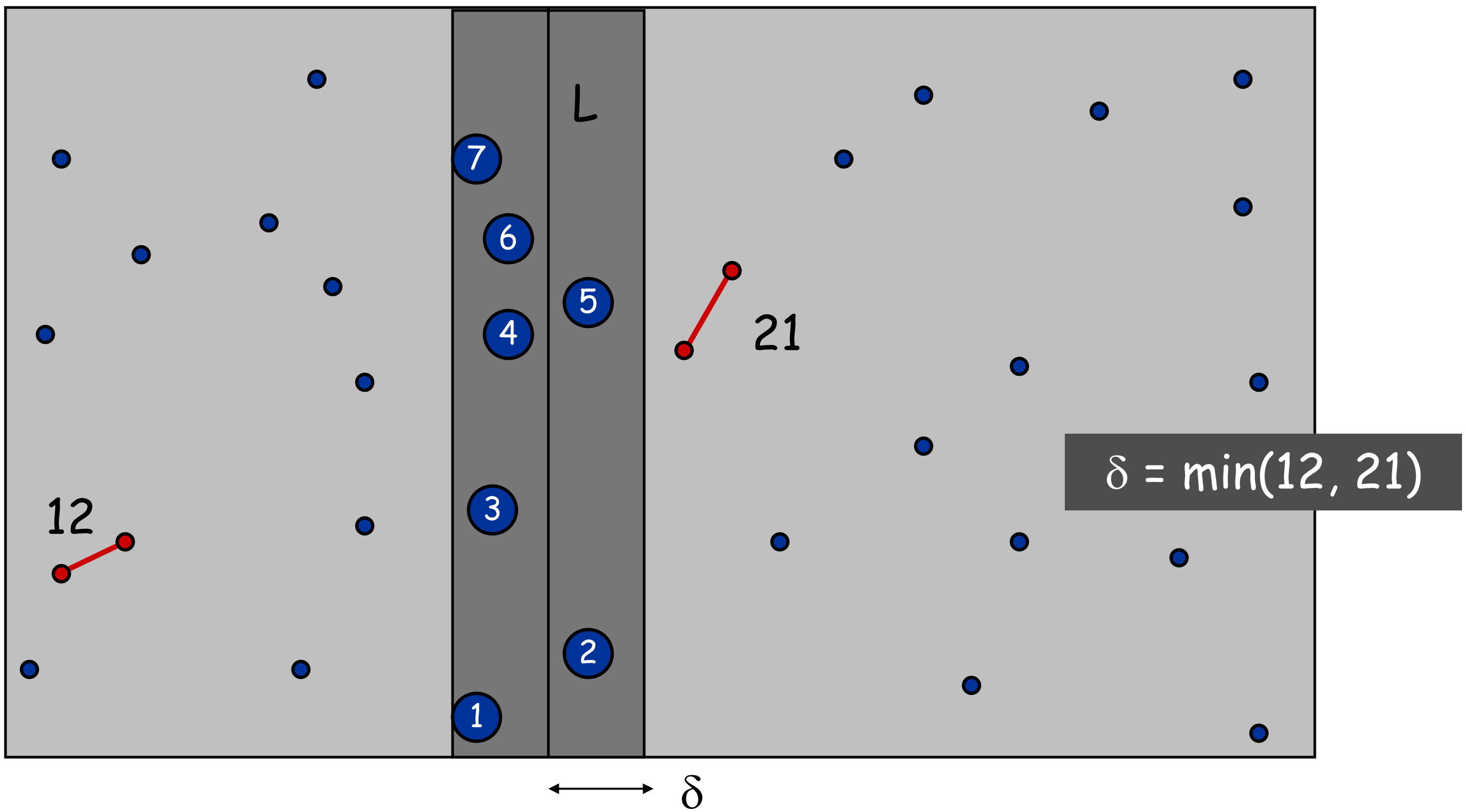
- Observation: only need to consider points within δ of line L.



Closest Pair of Points

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

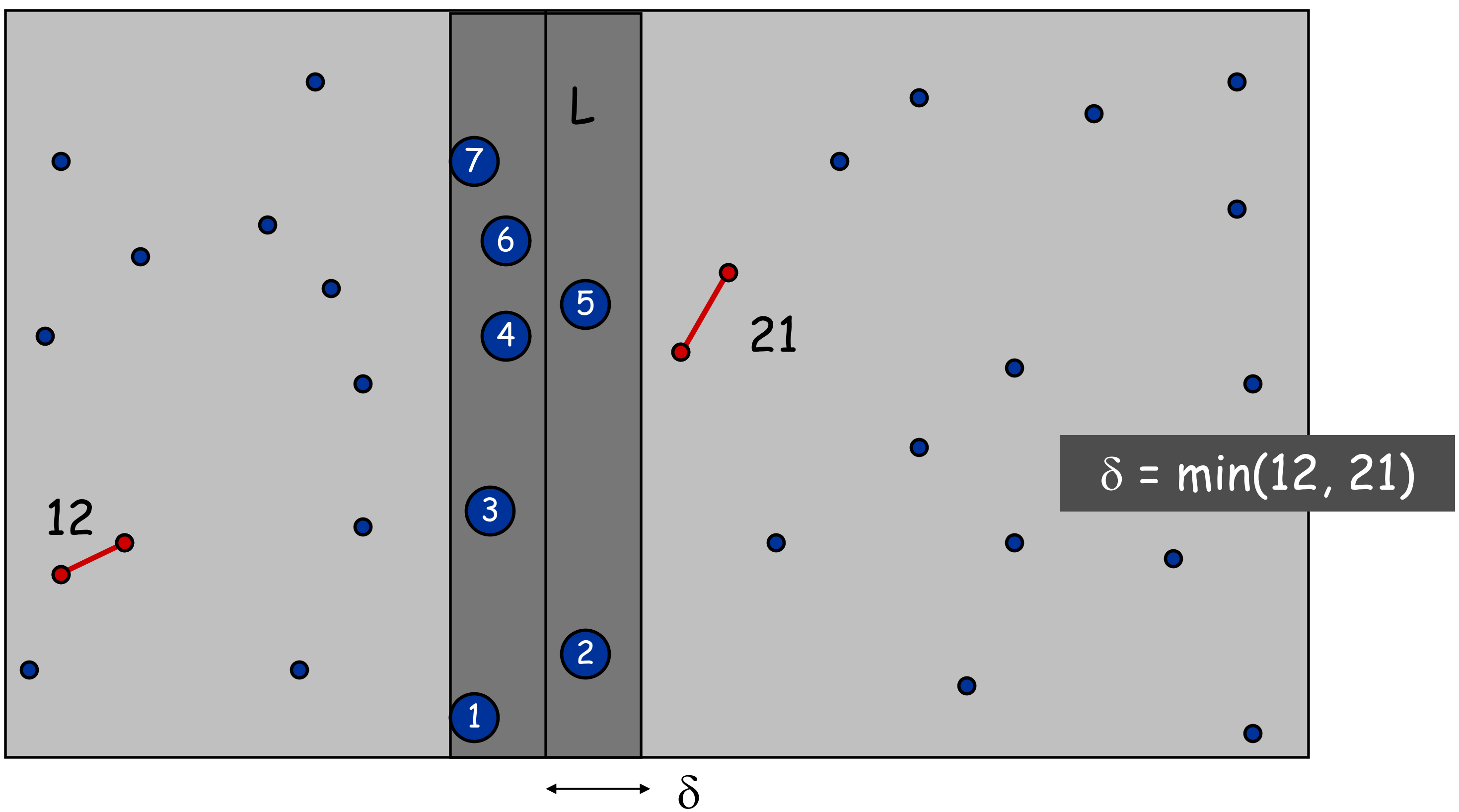
- Observation: only need to consider points within δ of line L.
- Sort points in 2δ-strip by their y coordinate.



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!



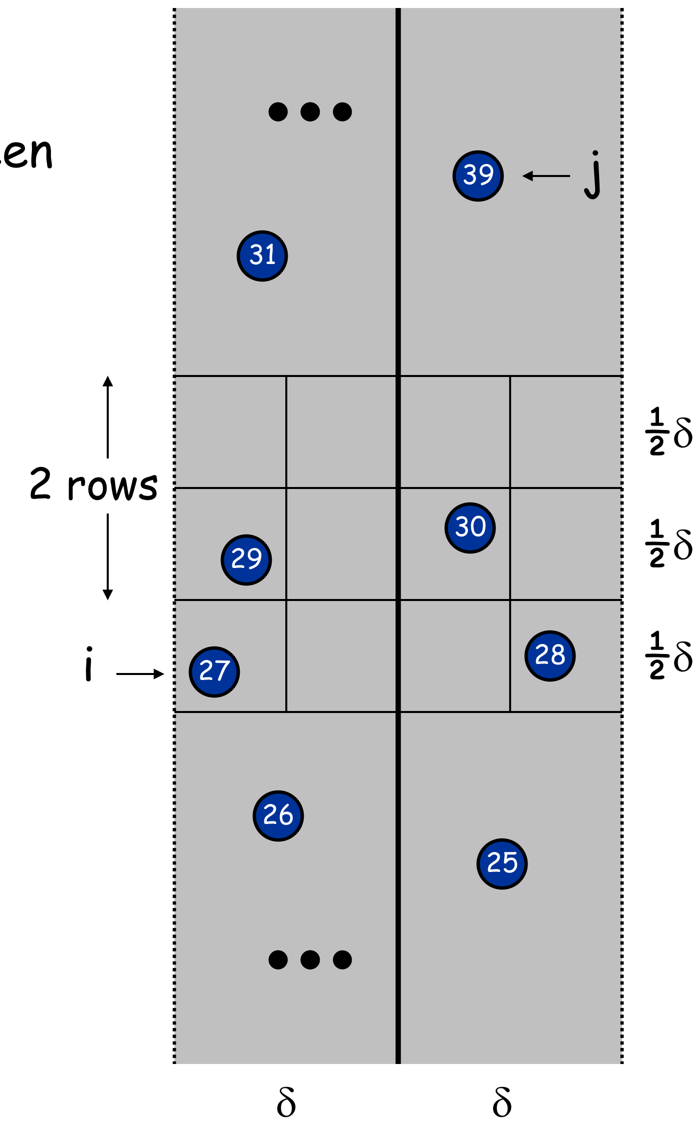
Closest Pair of Points

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.
(This is independent of δ calculated at each recursive call.)

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.
}
```

Compute separation line	$O(n \log n)$
Recursive calls	$2T(n / 2)$
Delete	$O(n)$
Sort	$O(n \log n)$
Scan	$O(n)$

Closest Pair of Points: Analysis

Running time.

$$T(n) \le 2T\big(n/2\big) + O(n \log n) \;\Rightarrow\; T(n) = O(n \log^2 n)$$

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.

$$T(n) \le 2T\big(n/2\big) + O(n) \;\Rightarrow\; T(n) = O(n \log n)$$

Q. Can we do better?

A. Yes, O(n) using randomized solution (Chapter 13)

Integer Multiplication

X times Y: half-and-half, but still $O(n^2)$

$$xy = (x_1 \cdot 2^{n/2} + x_0)(y_1 \cdot 2^{n/2} + y_0)$$
$$= x_1 y_1 \cdot 2^n + (x_1 y_0 + x_0 y_1) \cdot 2^{n/2} + x_0 y_0.$$

Complexity: $T(n) \leq 4T(n/2) + cn$

Reduce 4 calls to 3: $(x_1 + x_0)(y_1 + y_0) = x_1 y_1 + x_1 y_0 + x_0 y_1 + x_0 y_0$

New complexity:

$$T(n) \leq 3T(n/2) + cn$$

Hence,

$$O(n^{\log_2 3}) = O(n^{1.59})$$

```
Recursive-Multiply(x,y):
  Write x = x_1 · 2^{n/2} + x_0
        y = y_1 · 2^{n/2} + y_0
  Compute x_1 + x_0 and y_1 + y_0
  p = Recursive-Multiply(x_1 + x_0, y_1 + y_0)
  x_1y_1 = Recursive-Multiply(x_1, y_1)
  x_0y_0 = Recursive-Multiply(x_0, y_0)
  Return x_1y_1 · 2^n + (p − x_1y_1 − x_0y_0) · 2^{n/2} + x_0y_0
```
