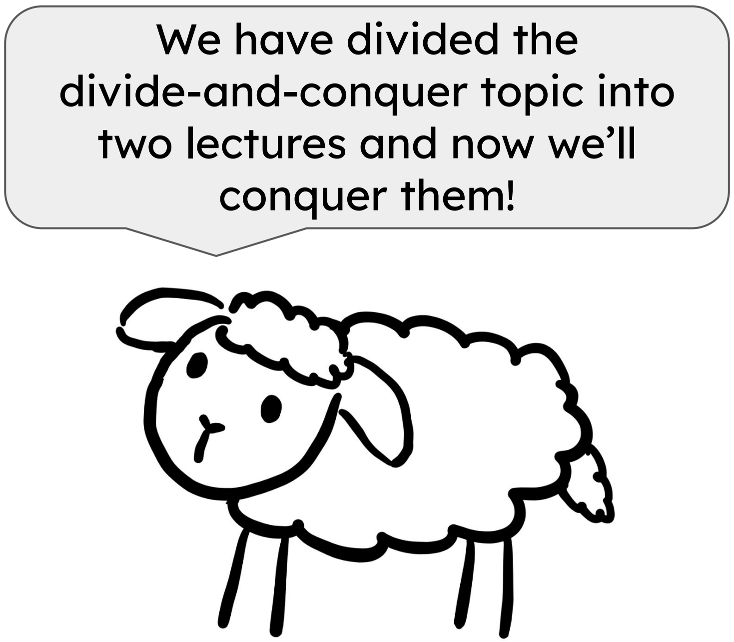
# Divide and Conquer Part 2



Another example of divide and conquer:

# Integer Multiplication

n digits

**Divide:**

| | $x_L$ | $x_R$ |
|---|---|---|
| $x =$ | $x_L$ | $x_R$ |
| $y =$ | $y_L$ | $y_R$ |

**Conquer:** $x \cdot y = (x_L \cdot 10^{n/2} + x_R)(y_L \cdot 10^{n/2} + y_R)$

$$= x_L y_L \cdot 10^n + (x_L y_R + x_R y_L) \cdot 10^{n/2} + x_R y_R$$

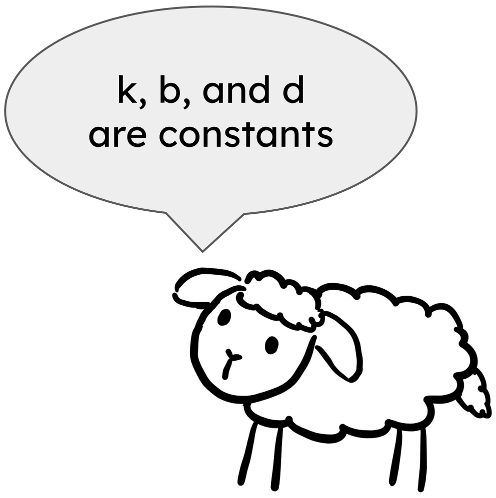
**Recurrence:**

# Solving Recurrences

## The Master Theorem

**Formally:** Consider the recurrence relation $T(n) = kT(n/b) + O(n^d)$, when $k, b > 1$. Then:

$$T(n) = \begin{cases} O(n^d) & \text{if } \left(k/b^d\right) < 1 \\ O(n^d \log n) & \text{if } \left(k/b^d\right) = 1 \\ O(n^{\log_b k}) & \text{if } \left(k/b^d\right) > 1 \end{cases}$$




$$T(1) = O(1)$$

(Earlier, Gauss used the same trick in a different context)

# Karatsuba's idea!

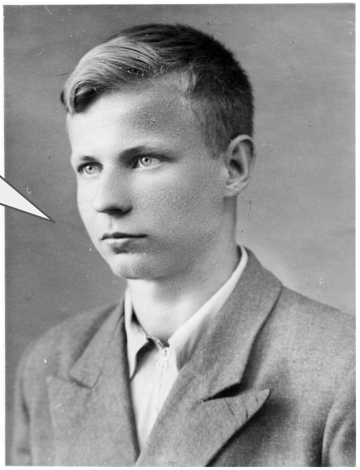
$O(n^2)$

Around 1956, the famous Soviet mathematician Andrey Kolmogorov conjectured that this is the *best possible way* to multiply two numbers together.

Just a few years later, Kolmogorov's conjecture was shown to be spectacularly wrong.

In 1960, Anatoly Karatsuba, a 23-year-old mathematics student in Russia, discovered a sneaky algebraic trick that reduces the number of multiplications needed.

We only need 3 recursive calls rather than 4!



# Karatsuba's idea!

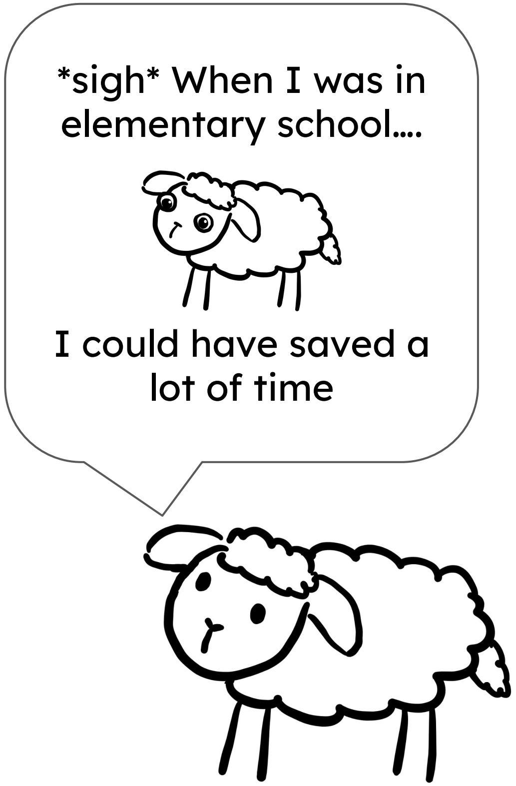
n digits

**Divide:**

| | | |
|---|---|---|
| $x =$ | $x_L$ | $x_R$ |
| $y =$ | $y_L$ | $y_R$ |

**Conquer:** $x \cdot y = x_L y_L \cdot 10^n + \cancel{(x_L y_R + x_R y_L)} \cdot 10^{n/2} + x_R y_R$

$(x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$



**Question:** It is possible to do even better than Karatsuba multiplication?

**Answer:** Yes - the best known result is $O(n \log n)$
[Harvey, van der Hoeven, 2019]

**Open problem:** Can this be improved to $O(n)$?

**Conjecture:** No (but we don't know)

Another example of divide and conquer:

# Closest Pair of Points

## Closest Pair Data Structures: Applications

The following algorithms and applications can be implemented efficiently using our new closest pair data structures, or involve closest pair computation as important subroutines.

- Dynamic minimum spanning trees
- Two-optimization heuristics in combinatorial optimization
- Straight skeletons and roof design
- Ray-intersection diagram
- Other collision detection applications
- Hierarchical clustering
- Traveling salesman heuristics
- Greedy matching
- Constructive induction
- Gröbner bases

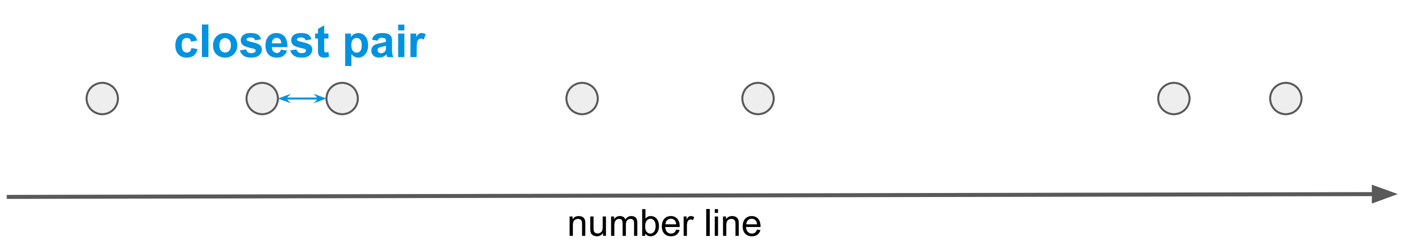
---

David Eppstein, Information & Computer Science, UC Irvine, .

# Warm-up: Closest Pair of Points in 1-D

- **Problem:** Given $n$ real numbers $x_1, x_2, \ldots, x_n$, find $i \neq j$ with the smallest $|x_i - x_j|$.

- **Solution:**
  - Sort the points.
  - Go over the list and compute the distances between the adjacent points.
  - Return the pair of adjacent points with the min distance.
- **Runtime:**

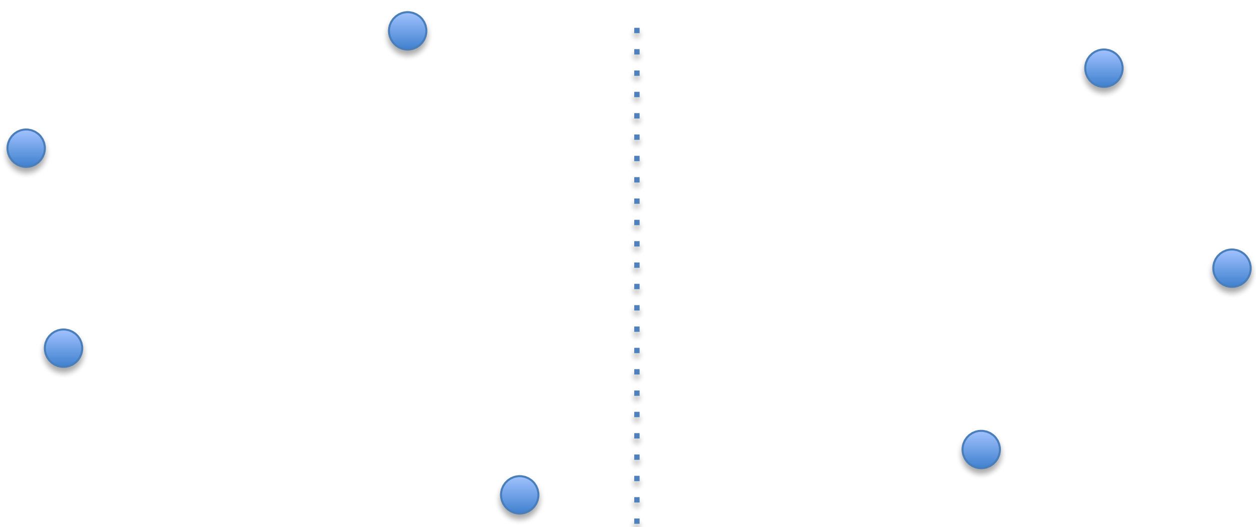


# Closest Pair of Points in 2-D

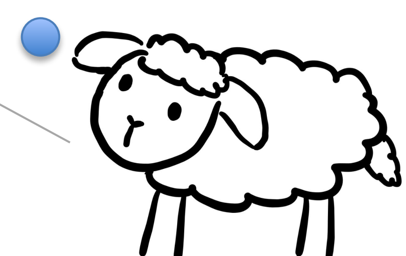
# Finding the Closest Pair of Points

**Input:** A list of n points in $R^2$: $[(x_1,y_1), (x_2,y_2), \ldots , (x_n,y_n)]$
**Output:** The closest pair of points
**Goal:** O(n log n) algorithm
**Working backwards:** We want a "mergesort" recurrence relation!

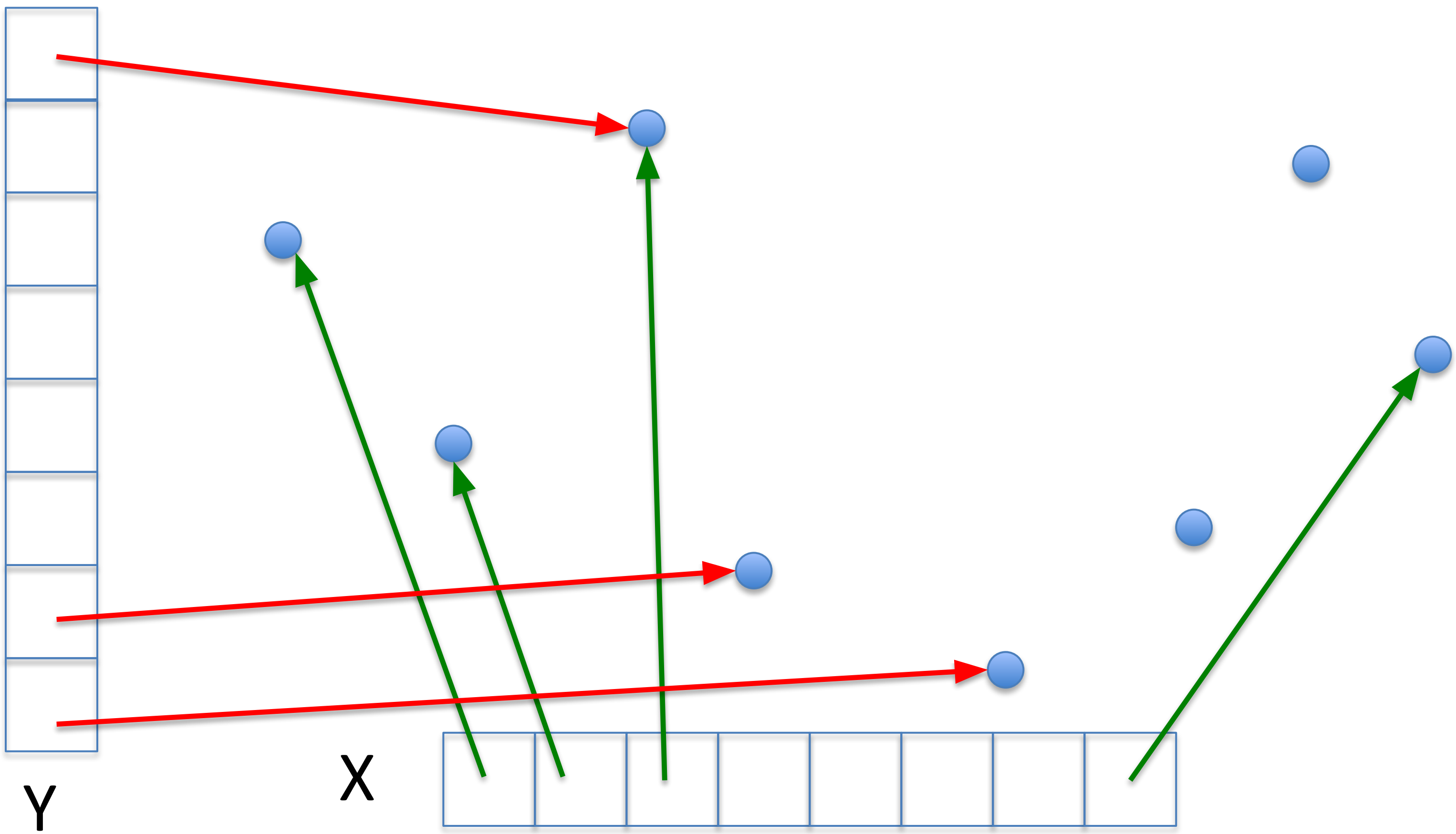
$T(n) = 2T(n/2)+O(n)$



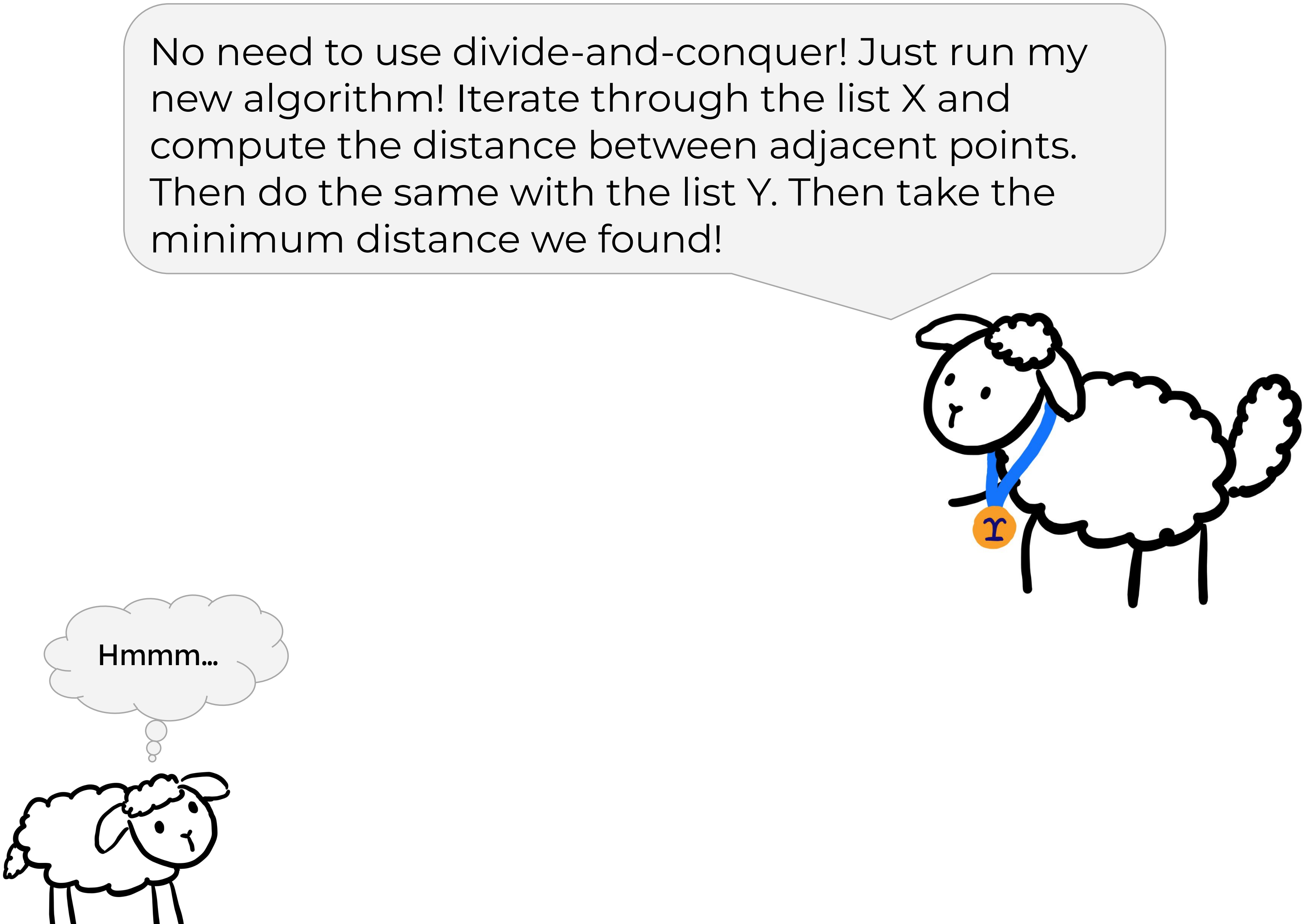
If we're willing to spend O(n log n) time, we may as well sort these points (just once)!



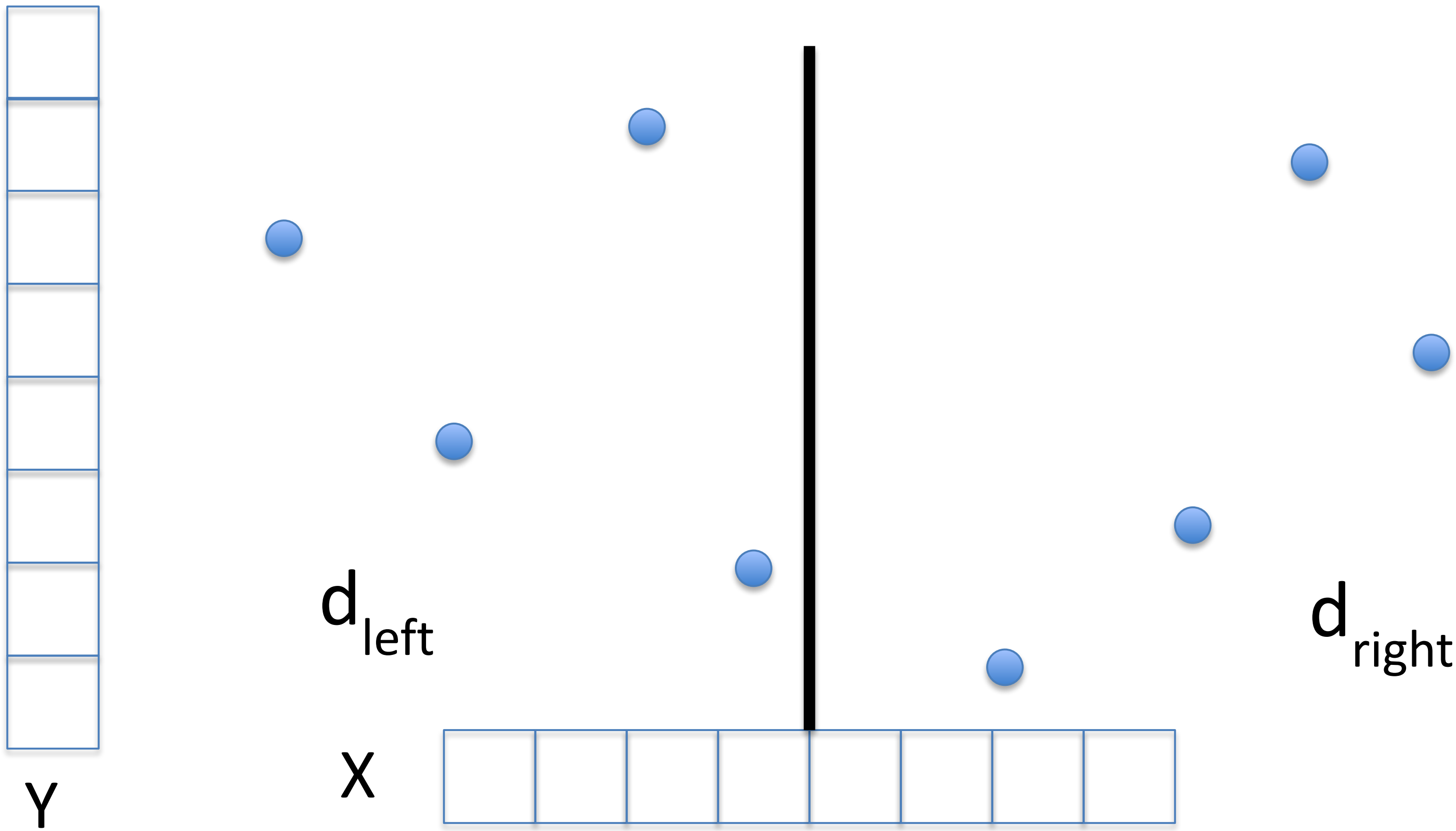
Build two copies of the input list: One sorted by x-coordinate and one sorted by y-coordinate



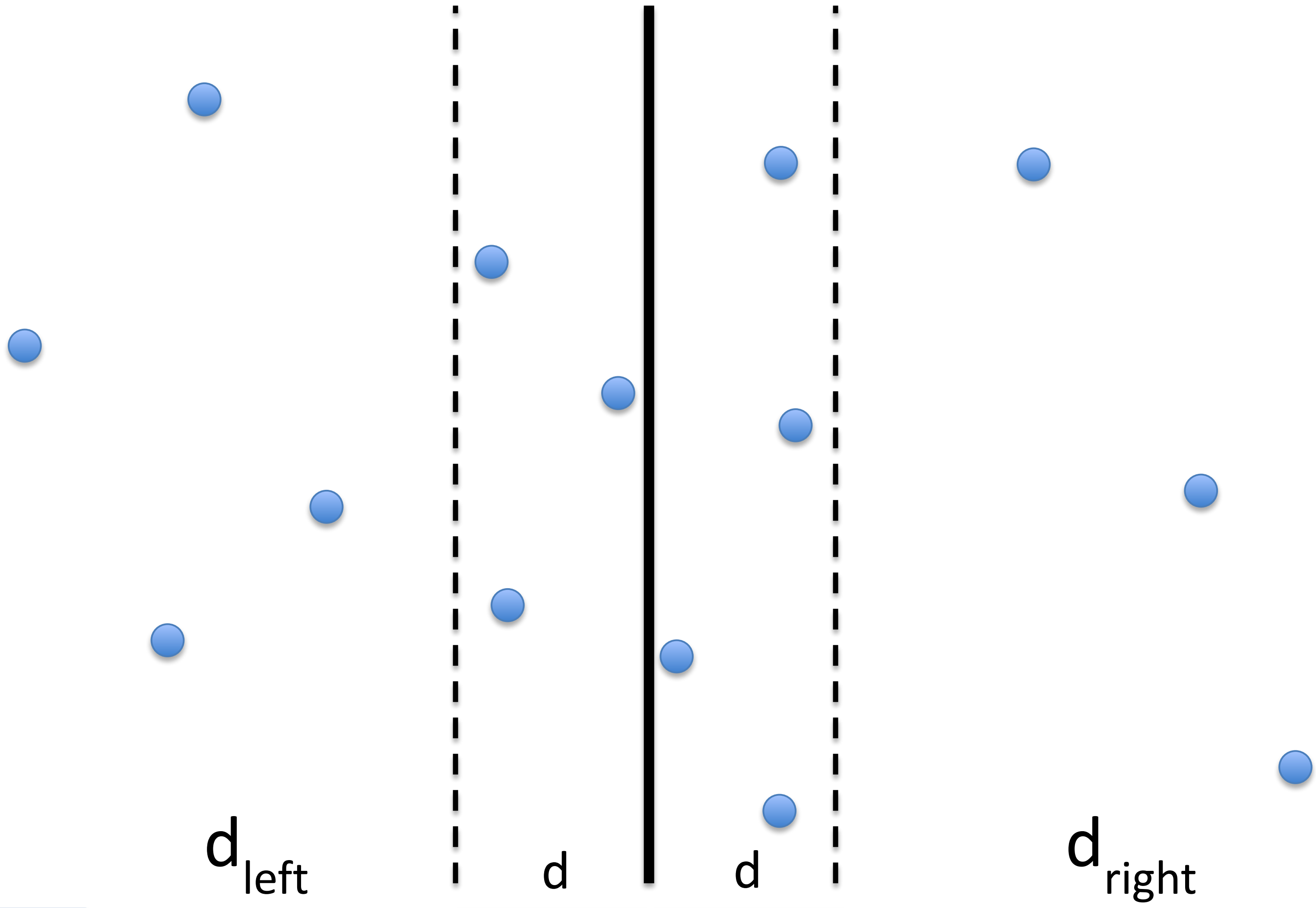

If we're willing to spend O(n log n) time, we may as well sort these points (just once)!


No need to use divide-and-conquer! Just run my new algorithm! Iterate through the list X and compute the distance between adjacent points. Then do the same with the list Y. Then take the minimum distance we found!
Hmmm...

# Divide-and-Conquer!

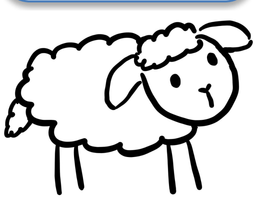


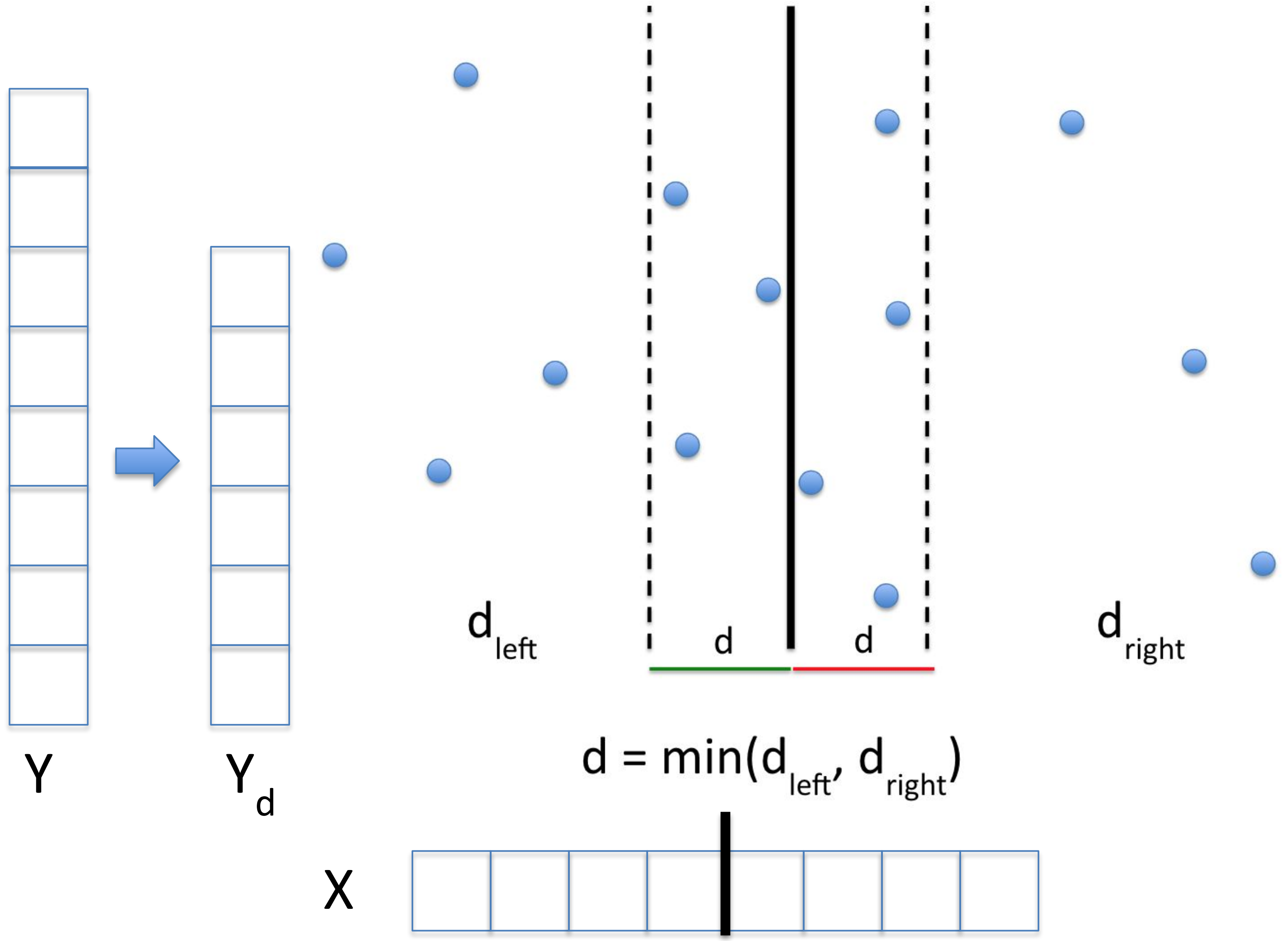

$d = \min(d_{left}, d_{right})$



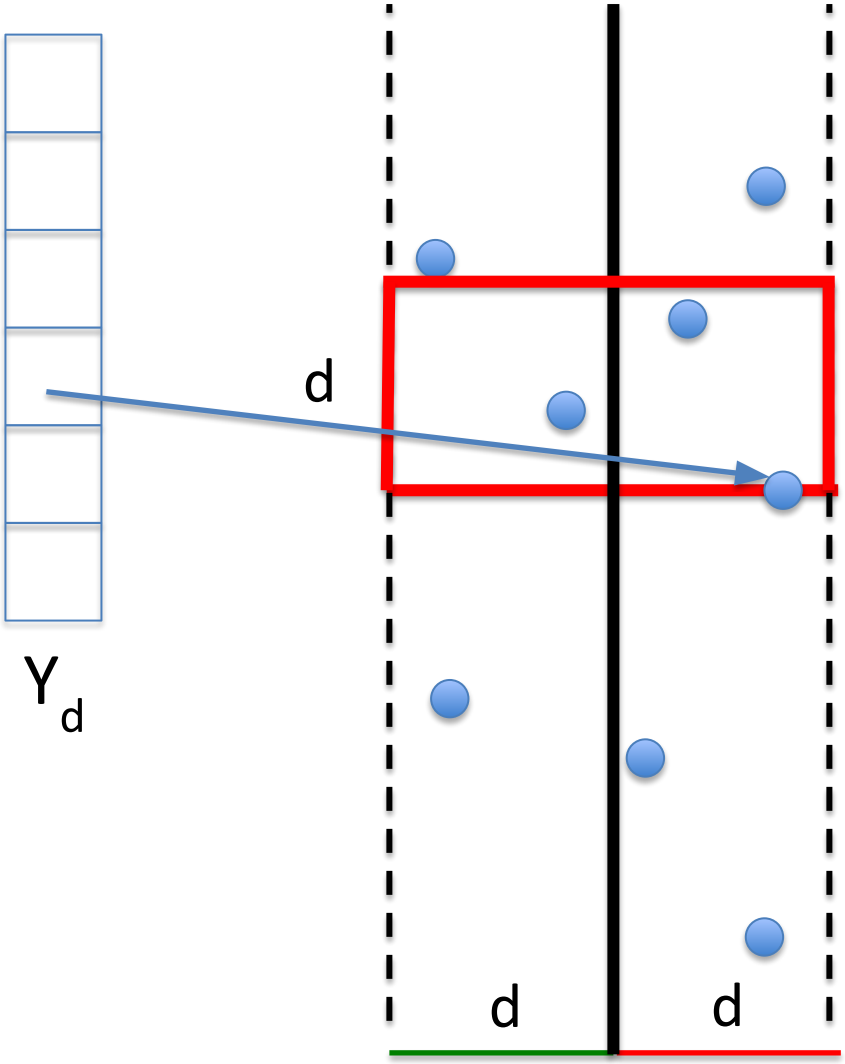

$$d = \min(d_{left}, d_{right})$$

Good news! (?)

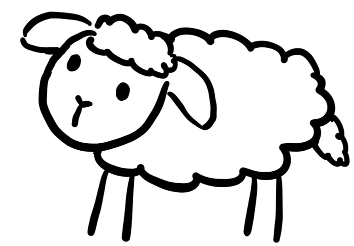


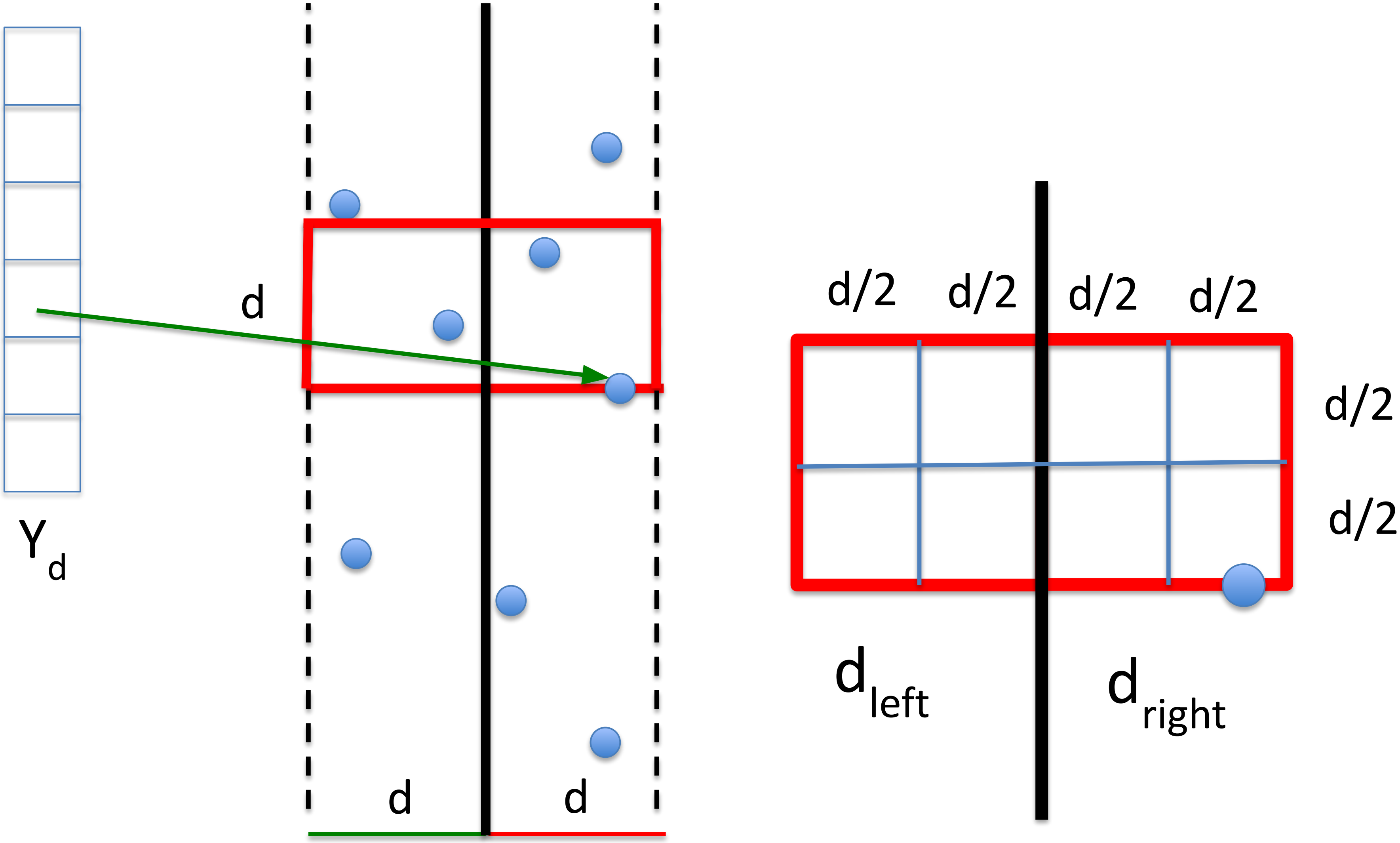


Y
$Y_d$
$d_{left}$
d
d
$d_{right}$
$d = \min(d_{left}, d_{right})$
X




Claim: There can't be too many points in this red d-by-2d rectangle? Why? How many points could there be?



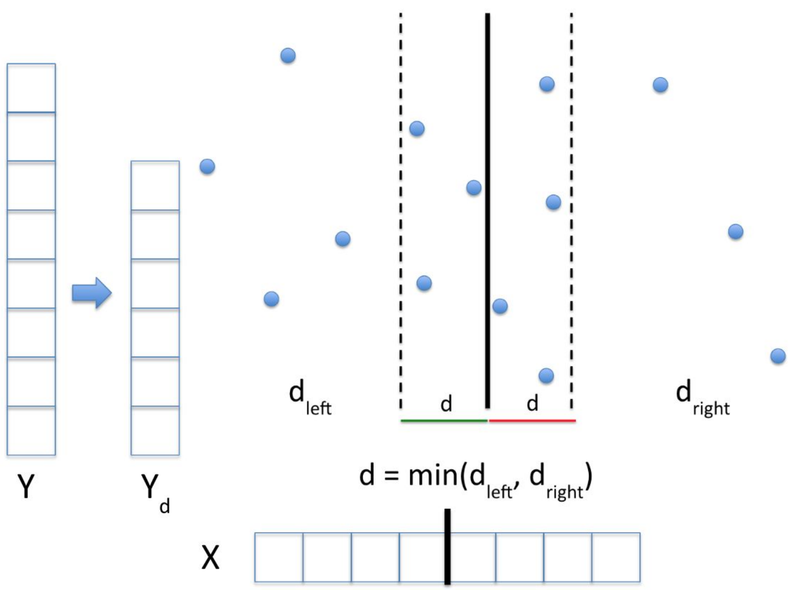



$$d = \min(d_{left}, d_{right})$$

# Total running time

- Sort once by x-coordinate: $O(n \log n)$
- Sort once by y-coordinate: $O(n \log n)$
- Recursive algorithm: $T(n) = 2T(n/2) + O(n)$

  $T(2) = O(1)$




Discovered in 1976 by
Jon Louis Bentley and Michael Ian Shamos