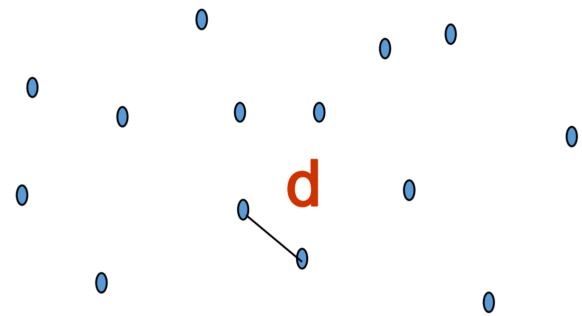
Closest Pair Problem : Brute Force

- Problem: Given a set of points, find the closest pair (measured in Euclidean distance)



- Brute-force method: $\theta(n^2)$.

BruteForceClosestPoints(*P*)   // *P* is list of points
***dmin* = ∞**
for i= 1 to n-1 do
for k = i +1 to n do
d = sqrt $((x_i - x_k)^2 + (y_i - y_k)^2)$
if *d* < *dmin* then
dmin* = *d*, pos1 = i, pos2 ← *k
return *pos*1, pos2

Closest Pair Problem : Brute Force

- A straightforward approach usually directly based on problem statement and definitions
- Motto : Just do it!
- Crude but often effective
- Examples already encountered:
    - Selection sort
    - Multiplying two n by n matrices
    - Computing a^n ($a > 0$, n a nonnegative integer) by multiplying a together n times

Closest Pair Problem : Brute Force

Pros and Cons of Brute Force

- Strengths:
  - Simplicity and Wide applicability
  - Yields reasonable algorithms for some important problems and standard algorithms for simple computational tasks
- Weaknesses:
  - Rarely produces efficient algorithms
  - Some brute force algorithms are in feasibly slow
  - Not as creative as some other design techniques

Closest Pair Problem : D&C method

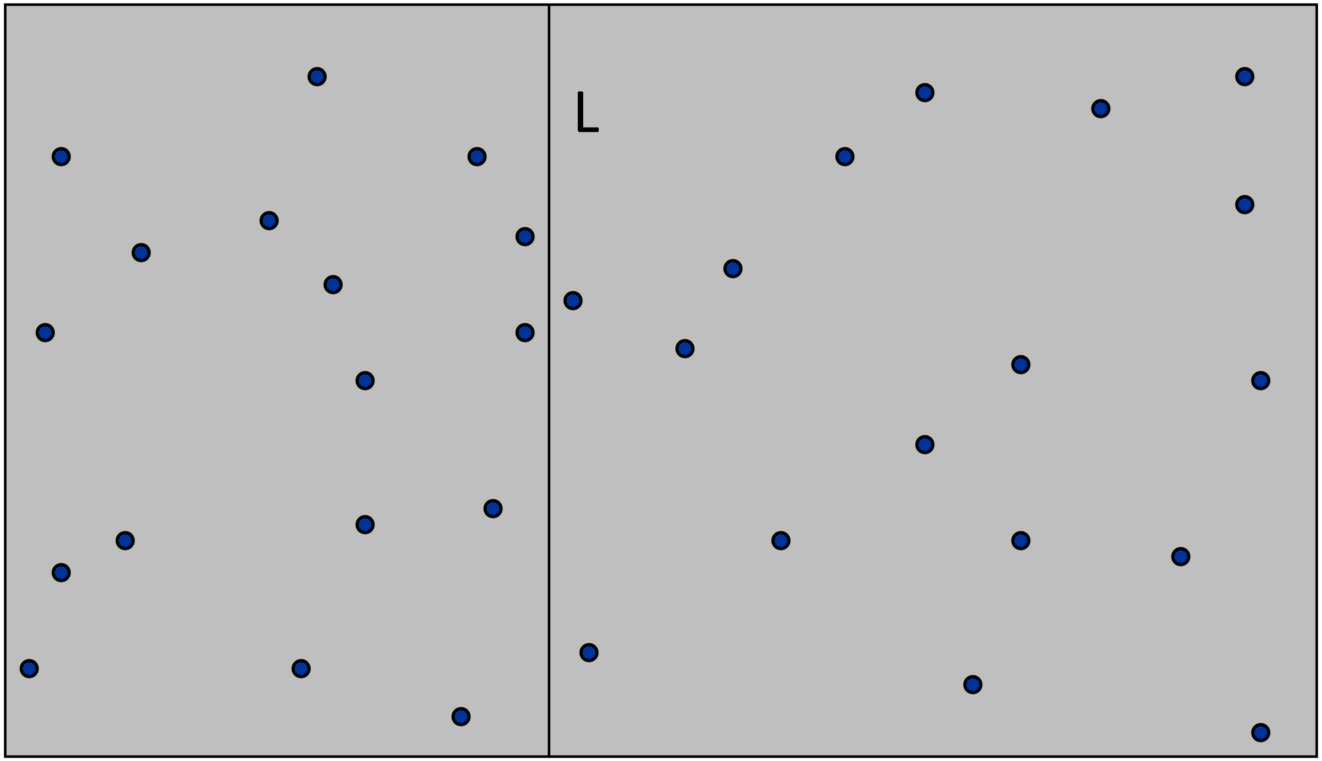
- Divide-and-conquer method:
  - Want to be lower than $O(n^2)$,

    $\Longrightarrow$ expect $O(n \log n)$.
  - Need $T(n)=2T(n/2)+O(n)$.
- How?
  - Divide : into 2 subsets (according to x-coordinate)
  - Conquer: recursively on each half.
  - Combine: select closer pair of the above.

One point from the left half and the other from the right may have closer distance.

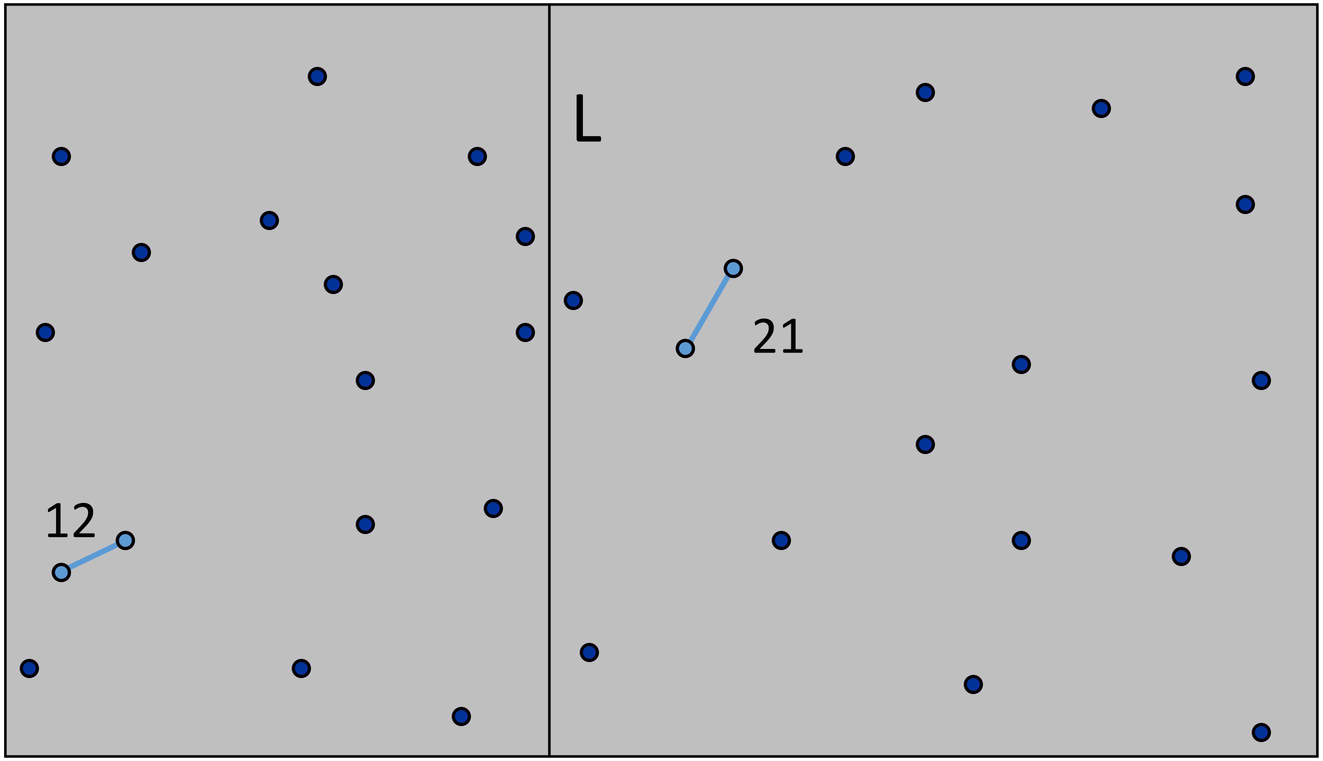
Closest Pair of Points

- Algorithm.
  - Divide: draw vertical line L so that roughly ½n points on each side.



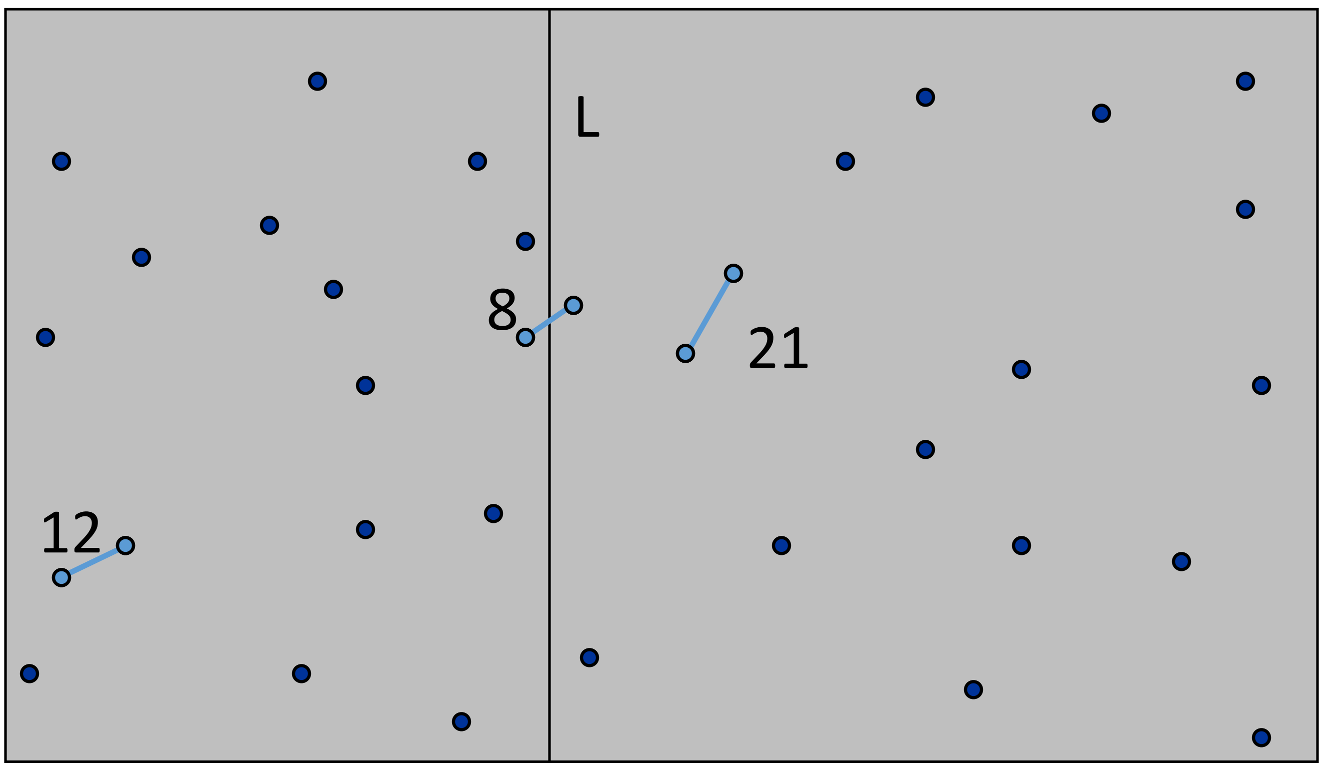
Closest Pair of Points

- Algorithm.
  - Divide: draw vertical line L so that roughly ½n points on each side.
  - Conquer: find closest pair in each side recursively.



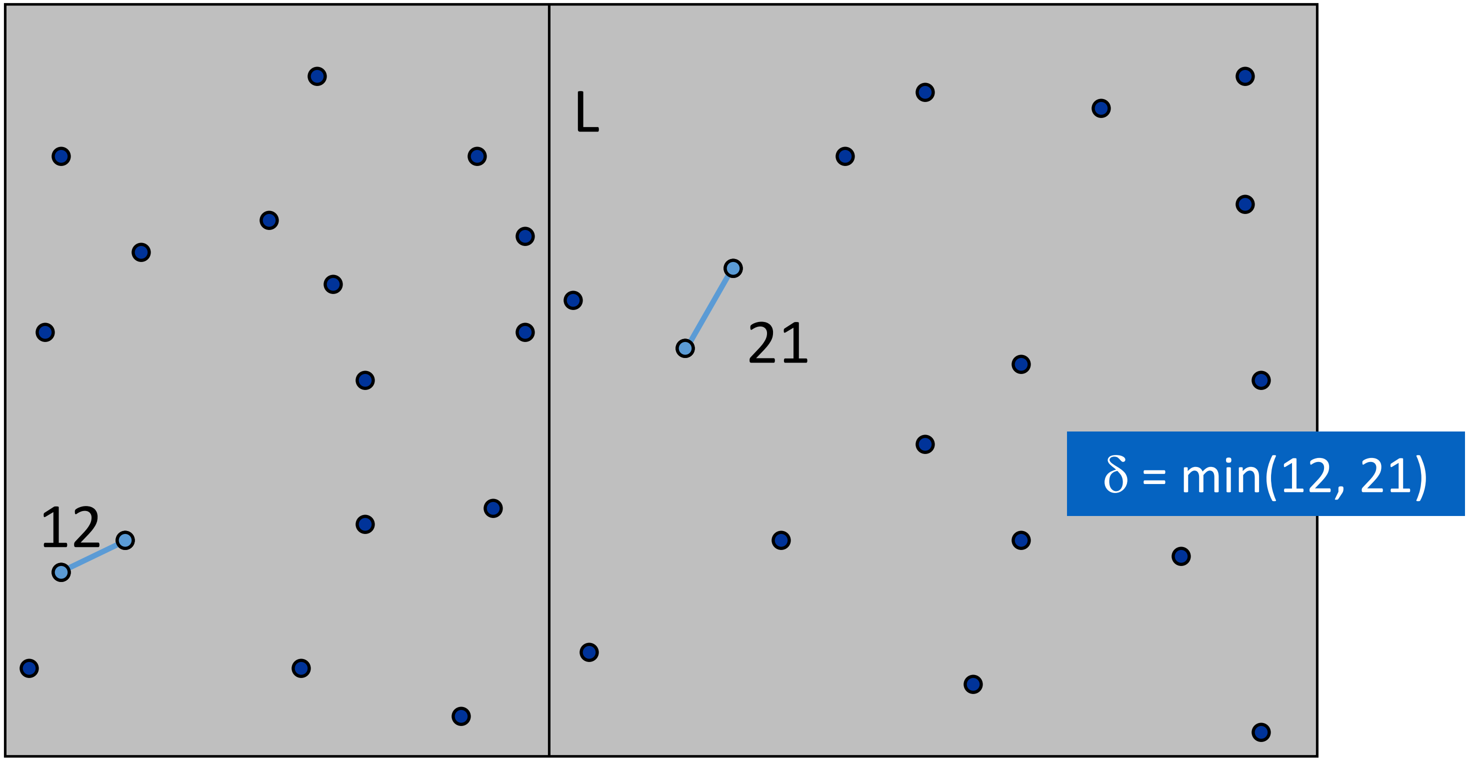
Closest Pair of Points

- Algorithm.
  - Divide: draw vertical line L so that roughly ½n points on each side.
  - Conquer: find closest pair in each side recursively. ← seems like $\Theta(n^2)$
  - Combine: find closest pair with one point in each side.
  - Return best of 3 solutions.



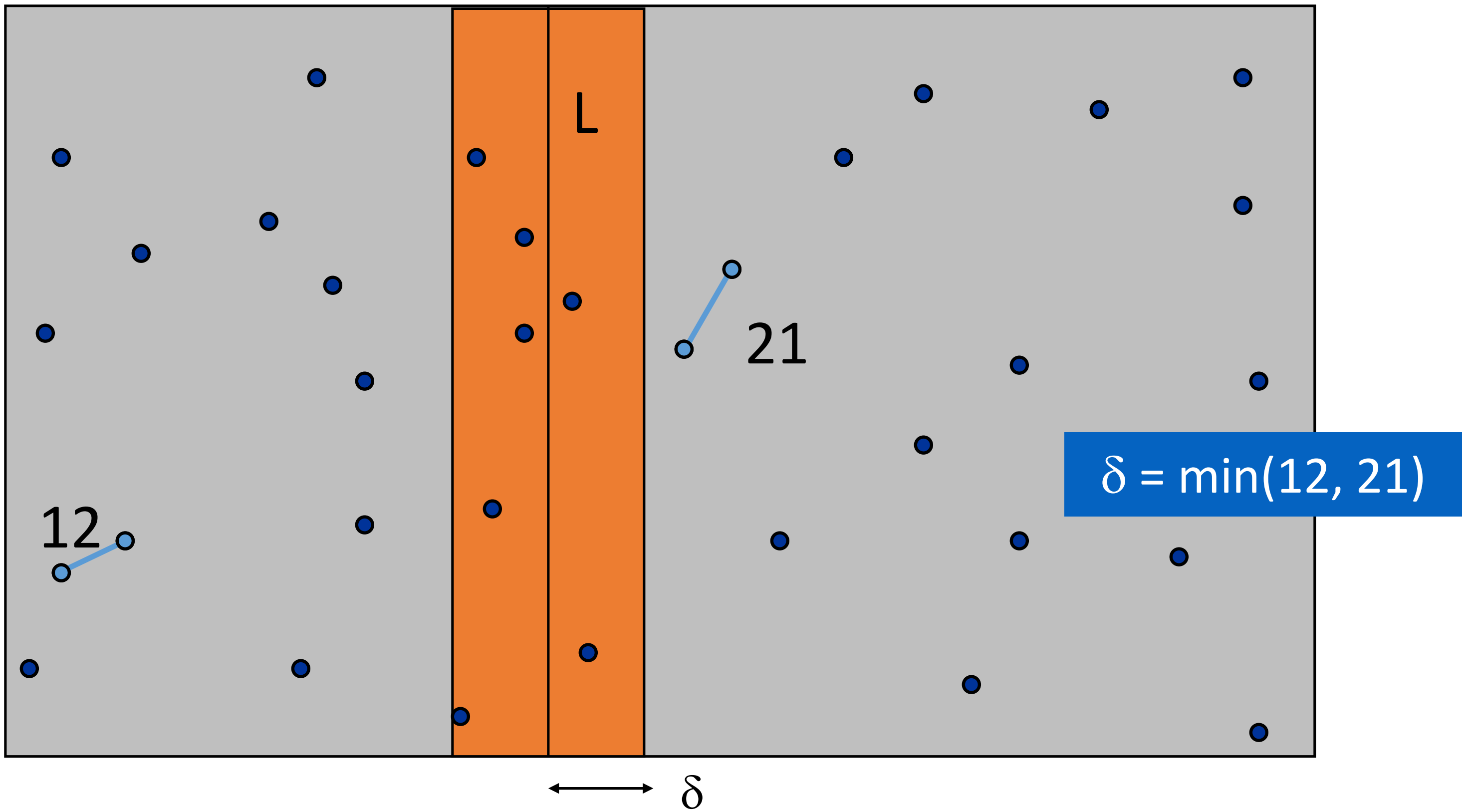
Closest Pair of Points

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



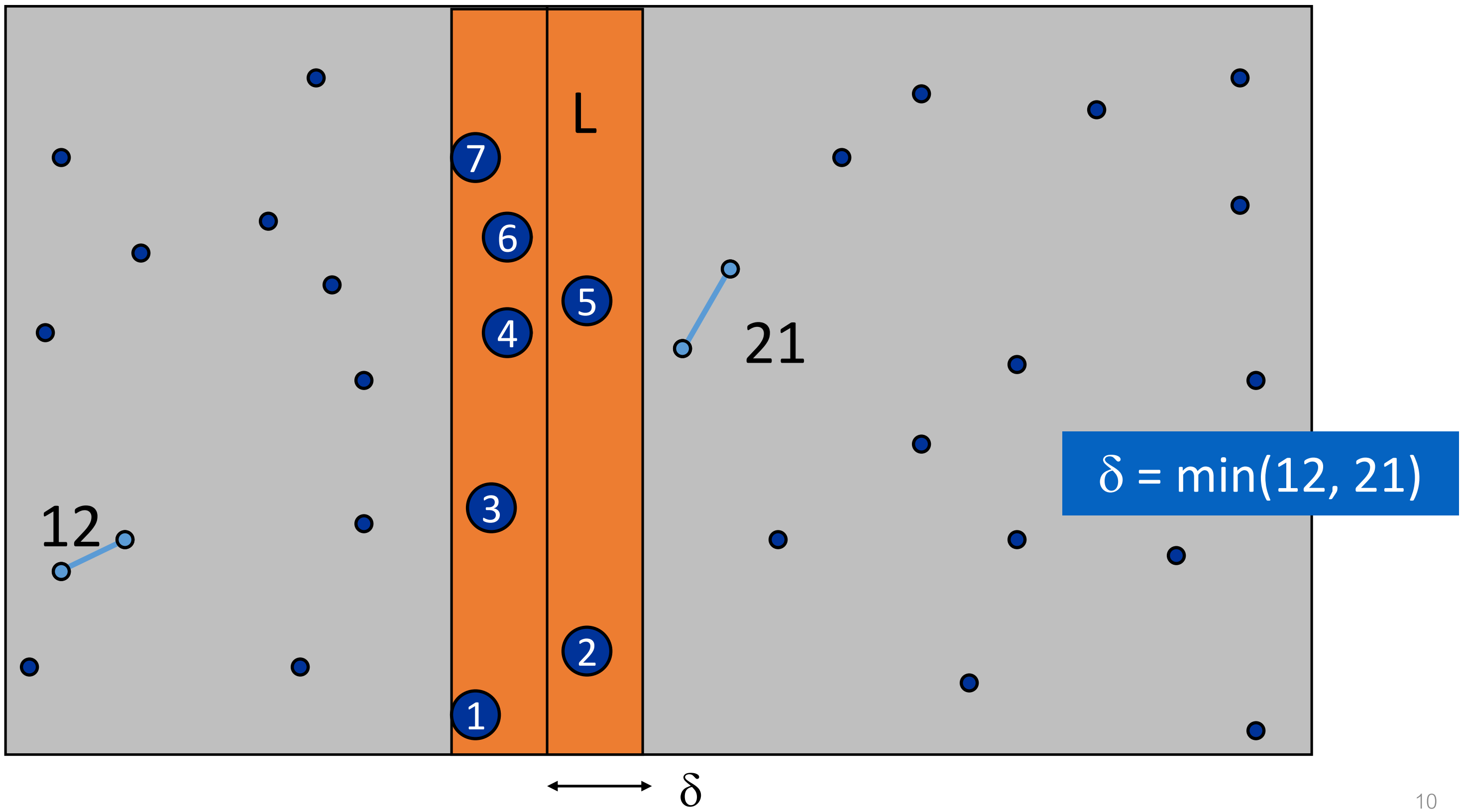
Closest Pair of Points

- Find closest pair with one point in each side, assuming that distance < δ.
  - Observation: only need to consider points within δ of line L.



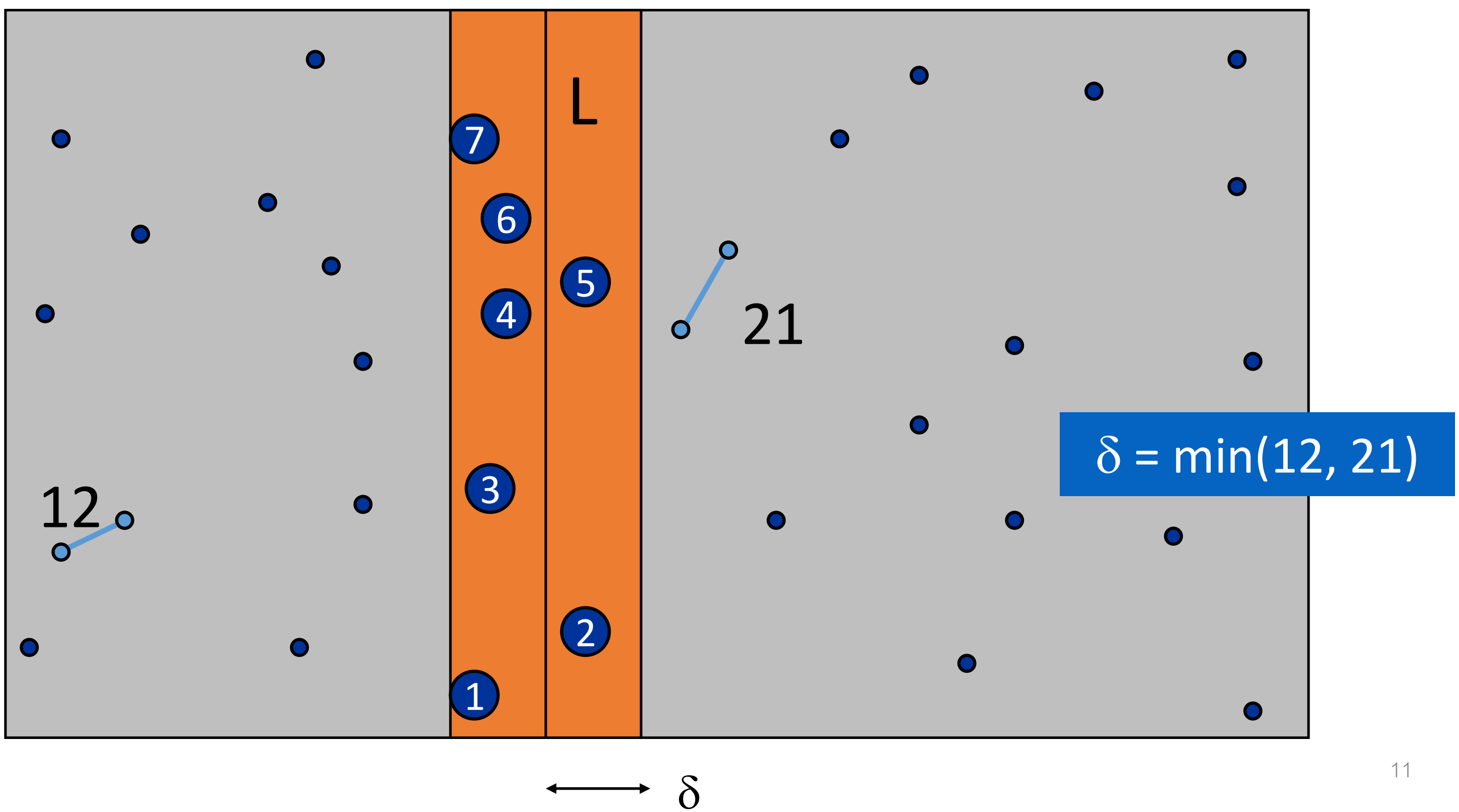
Closest Pair of Points

- 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!



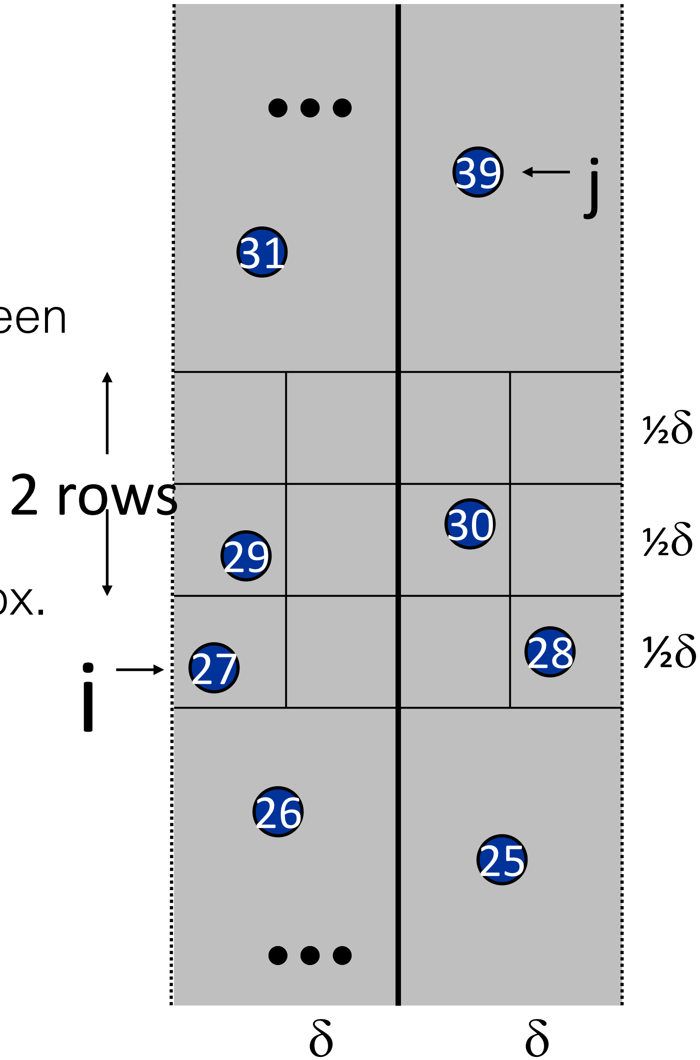
Closest Pair of Points

- 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    O(n log n)
   are on one side and half on the other side.

   δ1 = Closest-Pair(left half)                            2T(n / 2)
   δ2 = Closest-Pair(right half)
   δ  = min(δ1, δ2)

   Delete all points further than δ 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     O(n)
   each point and next 11 neighbors. If any of these
   distances is less than δ, update δ.

   return δ.
}
```

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.

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