Problem Formulation

- **Input**: set of points $P = \{p_1, \ldots, p_n\}$ where $p_i = (x_i, y_i)$
- **Assumption**: we can iterate over points in order of $x$- or $y$- coordinate in $O(n)$ time. Pre-generate data structures to support this in $O(n \log n)$ time.

Minimum Distance: Recursive Algorithm

1. Find vertical line $L$ to split points into sets $P_L$, $P_R$ of size $n/2$. $O(n)$
2. Recursively find minimum distance in $P_L$ and $P_R$.
   - $\delta_L = \text{minimum distance between } p, q \in P_L, p \neq q$. $T(n/2)$
   - $\delta_R = \text{same for } P_R$. $T(n/2)$
3. $\delta_M = \text{minimum distance between } p \in P_L, q \in P_R$. ??
4. Return $\min(\delta_L, \delta_R, \delta_M)$.

Naive Step 3 takes $\Omega(n^2)$ time. But if we do it in $O(n)$ time we get

\[ T(n) = 2T(n/2) + O(n) \implies T(n) = O(n \log n) \]
Making Step 3 Efficient

- **Goal:** Given $\delta_L$, $\delta_R$, compute $\min(\delta_L, \delta_R, \delta_M)$
- Let $\delta = \min(\delta_L, \delta_R)$. If $p \in P_L, q \in P_R$ are at least $\delta$ apart, they cannot be a closer pair, so we can ignore pair $(p, q)$.
- Let $S$ be the set of points within distance $\delta$ from $L$. We only need to consider pairs that are both in $S$.
- For a given point $p \in S$, how many other points in $S$ are within $\delta$ units of $p$ in the $y$ coordinate? **Intuition:** point in $S$ on either side of line can’t be too close to one another $\implies$ must “spread out” vertically

How to find closest pair with one point in each side?

**Def.** Let $s_i$ be the point in the $2\delta$-strip, with the $i$th smallest $y$-coordinate.

**Claim.** If $|j - i| > 7$, then the distance between $s_i$ and $s_j$ is at least $\delta$.

**Pf.**
- Consider the $2\delta$-by-$\delta$ rectangle $R$ in strip whose min $y$-coordinate is $y$-coordinate of $s_i$.
- Distance between $s_i$ and any point $s_j$ above $R$ is $\geq \delta$.
- Subdivide $R$ into 8 squares.
- At most 1 point per square.
- At most 7 other points can be in $R$. $

Wrap-Up

- **Step 3 is $O(n)$:** iterate in order of $y$ coordinate and compare each point to constant number of neighbors.
- $\implies O(n \log n)$ overall.
- **Intuition:** we reduced Step 3 (almost) to 1D closest-pair
  - Iterate, compare each point to next $k$ points (instead of 1)
  - The set $S$ is “nearly one-dimensional”. Points cannot be packed too tightly, because pairs on each side have to be at least $\delta$ apart.
  - For $d > 2$ dimensions, there is a divide and conquer algorithm where the “combine” step (i.e., Step 3) solves a closest pair problem in $d - 1$ dimensions
Closest Pair in $d$ Dimensions

Board work
Solve recurrence

$$T(n, d) = 2T(n/2, d) + T(n, d - 1)$$

Base case $T(n, 2) = \Theta(n \log n)$
Solution: $T(n, d) = \Theta(n \log^{d-1} n)$