The following three sections introduce the divide-and-conquer algorithm for closest pair. Why? I want to show you an interesting, nontrivial algorithm and how we analyze it.

We compute distance only. Of course, you can save points whenever min distance is updated, just as for naive algorithm.

1 Algorithm Part 1: Preprocessing

- Recall that $P$ is input array of $n$ points. [Sketch five points in space as shown at right]

- Create two sorted arrays of references to points in $P$. Note that arrays refer to same points, just in different orders.
  - $ptsByX$ enumerates points of $P$ in increasing order by $x$.
  - $ptsByY$ enumerates points of $P$ in increasing order by $y$.

[Embellish the five points with $ptsByX$, $ptsByY$ illustrations:]

- Algorithm $ClosestPair$ takes sorted arrays $ptsByX$, $ptsByY$, and input size $n$.

2 Algorithm Part 2: Divide-and-Conquer Skeleton

The divide-and-conquer strategy:

- split large problem into smaller parts (divide)
- solve the smaller parts recursively (conquer) (maintain invariant: parts must be sorted like original problem!)
- combine smaller solutions (**combine**)
- Can be much faster than solving entire problem at once

**ClosestPair**(*ptsByX*, *ptsByY*, *n*)

  if *n* = 1  
  \[ \text{return } \infty \]

  if *n* = 2  
  \[ \text{return } \text{distance}(ptsByX[0], ptsByX[1]) \]

  mid ← \lceil n/2 \rceil − 1  \hspace{1cm} \triangleright \text{divide into two subproblems}
  
  copy *ptsByX*[0...mid] into new array *XL* \textit{in x order}.
  
  copy *ptsByX*[mid+1...n−1] into new array *XR* \textit{in x order}.

  copy *ptsByY* into arrays *YL* and *YR* \textit{in y order}, s.t.
  
  *XL* and *YL* refer to same points, as do *XR* and *YR*

  distL ← **ClosestPair**(*XL*, *YL*, \lceil n/2 \rceil)  \hspace{1cm} \triangleright \text{conquer}

  distR ← **ClosestPair**(*XR*, *YR*, \lfloor n/2 \rfloor)

  \[ \text{return } \text{Combine}(ptsByY, ptsByX[mid], n, \text{min}(\text{distL}, \text{distR})) \]
3 Algorithm Part 3: Combine Step

To combine, must consider pairs of points that cross the dividing line in \( x \).

\[
\text{COMBINE}(\text{ptsByY}, \text{midPoint}, n, \text{lrDist})
\]

construct array \( \text{yStrip} \), in increasing \( y \) order, of all points \( p \) in
\[
\text{ptsByY s.t. } |p.x - \text{midPoint}.x| < \text{lrDist}
\]

\[
\text{minDist} \leftarrow \text{lrDist}
\]

\[
\text{for } j \text{ in } 0 \ldots \text{yStrip.length} - 2 \text{ do}
\]

\[
k \leftarrow j + 1
\]

\[
\text{while } k \leq \text{yStrip.length} - 1 \text{ and } \text{yStrip}[k].y - \text{yStrip}[j].y < \text{lrDist} \text{ do}
\]

\[
d \leftarrow \text{distance(}\text{yStrip}[j], \text{yStrip}[k])
\]

\[
\text{minDist} \leftarrow \min(\text{minDist}, d)
\]

\[
k++
\]

\[
\text{return } \text{minDist}
\]

[Illustrate difficult first step on diagram, including the center STRIP]

4 Correctness

In divide-and-conquer algorithms, correctness proofs are generally by induction on input size \( n \).

- **Base case:** trivially correct for \( n = 1, 2 \)

- **Inductive case:** Assume distL and distR are min. pairwise dists between points on either side of partition (size \(< n \)).

- If both points in closest pair are on same side, no pairs checked by \( \text{COMBINE} \) can change \( \text{minDist} \), and we are done.
• If points \((p, q)\) in closest pair are on opposite sides . . .
  
  – \(p\) and \(q\) at distance less than \(lrDist\)
  – \(p.x\) and \(q.x\) must be within \(lrDist\) of each other; hence, each is within \(lrDist\) of partition line in \(x\)
  – Moreover, \(p.y\) and \(q.y\) must be with \(lrDist\) of each other, so are found by \texttt{while} loop.

• Hence, closest pair is always found for size \(n\). QED

5 Cost, Part 1

What is worst-case running time of \texttt{ClosestPair} on inputs of size \(n\)? Let’s try \textit{statement counting} (without being too careful about constants):

[Do following counts and defns on overhead of algorithm.]

• call \(T(n)\) the running time of the algorithm on input of size \(n\)
• base case takes constant time \(c_0\)
• creating \(XL, YL, XR, YR\) takes time \(c_1n\) (DO NOT SORT!)
• creating array \(yStrip\) takes time \(c_2n\) (DO NOT SORT!)
• \textit{what about recursive calls?} Let’s write costs implicitly: \(T(\lceil n/2 \rceil)\) and \(T(\lfloor n/2 \rfloor)\)
• \textit{what about \texttt{COMBINE}?} Outer loop statements are \(c_3n + c_4\).
• \textit{Inner \texttt{while} loop?} Naively, seems it could run \(yStrip.length - j - 1\) times????

Now for the cool part!
(back to board)

6 Cost, Part 2

Claim: inner loop of \texttt{COMBINE} never runs more than seven times.

• Consider loop execution for any point \(yStrip[j]\).
• Each loop iteration handles a distinct point \(yStrip[k]\) inside a box of size 2 \(lrDist\) wide by \(lrDist\) high
• Any two points on same side of partition are at least \(lrDist\) apart!

\textit{Lemma:} (geometry) you can’t fit five points in a \(\delta \times \delta\) box \textit{and} have every pair be at distance at least \(\delta\).

[Draw box diagram on board:]

4
• “Obvious” that two points in a $\delta/2$ by $\delta/2$ box are always at distance less than $\delta$.

• Divide box into four quarters, and throw five points into box. By pigeonhole principle, some quarter contains two points.

• Hence, not all pairs in box at distance $\geq \delta$. QED

finish the proof

• Left and right halves of big box are lrDist by lrDist

• Hence, each half contains at most four points (else some pair on same side would be closer than lrDist).

• Conclude that box contains only eight points, including yStrip[$j$]. QED

7 Cost, Part 3

OK, we’ve filled in missing inner loop time. Conclude that...

$$T(n) = \begin{cases} 
  c_0 & \text{if } n \leq 2 \\
  T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + cn + d & \text{if } n > 2 
\end{cases}$$

• Assume $n$ is power of two for simplicity. Also, increase $c$ until linear term dominates constant term. This can only increase our time estimate, so no harm done.

• This is a recurrence for time $T(n)$: a definition of $T(n)$ in terms of $T(n')$, for $n' < n$.

• How do we solve this recurrence to find $T(n)$ in terms of $n$? Detailed discussion later, but here’s a good graphical method: the recursion tree.
1. How many levels in tree? Each time, we divide $n$ by 2, so to reach 2 (the base case), we need $\log_2 n$ levels.

2. On $k$th level (root has $k = 0$), input size is $n/2^k$, so we do $c'n/2^k$ work per node (besides recurring). But there are $2^k$ nodes on level $k$, so . . .

3. we do $c'n$ total work per level.

4. Conclude that total work done by ClosestPair in worst case is at most $c'n \times \log_2 n$. Remember those graphs? Which algorithm is better?
```python

ClosestPair(ptsByX, ptsByY, n)
    if n = 1
        return \infty
    if n = 2
        return distance(ptsByX[0], ptsByX[1])

mid \leftarrow \lceil n/2 \rceil - 1
     copy ptsByX[0 \ldots mid] into new array \textit{XL in x order}.
     copy ptsByX[mid+1 \ldots n − 1] into new array \textit{XR in x order}.

     copy ptsByY into arrays \textit{YL and YR in y order}, s.t.
     \textit{XL and YL refer to same points, as do XR and YR}.

    distL \leftarrow \textit{ClosestPair}(XL, YL, \lceil n/2 \rceil)
    distR \leftarrow \textit{ClosestPair}(XR, YR, \lfloor n/2 \rfloor)

midPoint \leftarrow ptsByX[mid]
lrDist \leftarrow \textit{min}(distL, distR)
     Construct array \textit{yStrip}, in increasing y order, of all
     points p in ptsByY s.t. |p.x − mid.x| < lrDist

    minDist \leftarrow lrDist
    \textbf{for } j \textbf{ in } 0 \ldots \textit{yStrip.length − 2 do}
    k \leftarrow j + 1
    \textbf{while } k \le \textit{yStrip.length − 1 and}
     \textit{yStrip[k].y − yStrip[j].y < lrDist do}
    d \leftarrow \textit{distance(yStrip[j], yStrip[k])}
    minDist \leftarrow \textit{min}(minDist, d)
    k++
    \textbf{return } \textit{minDist}
```