Divide-and-Conquer Algorithms

1. Divide - divide problem into subproblems (smaller input for same problem)

2. Recursively solve subproblems (or directly solve when input reaches a termination condition)

3. Combine - use subproblem solutions to solve given problem
Mergesort $a[0], ..., a[n-1]$

1. **Divide** - split array in half
   - Input (problem) sort $a[p] ... a[r]$
   - Subproblem $q = \lfloor (p+r)/2 \rfloor$
   - Sort $a[p] ... a[q]$ and $a[q+1] ... a[r]$

2. Recursively sort two subarrays

3. Merge two sorted subarrays
Example Execution

6 4 3 8 1 7 2 5

3 4 6 8 1 2 5 7

temp array

1 2 3 4 5 6 7 8

This takes linear time (in site of resulting subarray)
Closest pair of points

Divide, split into left half and a right half of \( \sim n/2 \) points each.

Recursively find closest pair on each of these two subproblems.

Combine - use subproblem solutions to find closest pair in given point set.
Consider each point in Strip looking upward.

For \( i = 0 \) to \( \text{numInStrip} - 1 \) do

\[ j = i + 1 \]

While \( j < \text{numInStrip} \) and \( y \text{Strip}(j) \cdot y - y \text{Strip}(i) \cdot y < d \)

\[ d = \min(d, \text{distance point } i \text{ to point } j) \]

\[ j++ \]

List of closest pairs found so far.
Closest pair of points (cont.)

Combine step in more detail: \( d = \min(d_L, d_R) \)

**Step 1**

Construct \( y \)-Strip to ref all points in order of \( pts \) by \( y \)-coord with \( x \)-coord \( > x_R - d \) or \( < x_L + d \)

**Step 2**

Find any pair with \( y \)-Strip with distance \( < \Delta \) (one from each side)
Example Execution

\[ d = \min(d_L, d_R) \]

pts By Y

pts By X

left subproblem

right subproblem

\[ d_R = d \]
don't need to consider with any point from right half

no point here is part of a closest pair with left half
Correctness

1. If closest pair of points are both on left we find in recursive call for left. Same for right.

2.a) We've argued that any pair of points not in y-strip need only be considered by 1.

2.b) We've argued that any pair in y-strip that could have dist < d is considered.
Correctness (cont.)

See the handout for the details of the correctness argument.

I've discussed the components when describing the algorithm.
$T(n) = \text{time complexity for } n \text{ points}$

Analysis of Asymptotic Time Complexity

**Divide** can be done in linear time. You must create points by $X_{\text{Left}}, X_{\text{Right}}, Y_{\text{Left}}, Y_{\text{Right}}$ carefully.

**Recursive calls**

$2T(n/2)$

**Combine** - argue this is linear
Time complexity of Combine step $\leq 2d$

Consider at most 5 points in "while" loop for "point $i".
Expressing Time Complexity using Recurrence equation

\[ T(n) = C_1 \cdot n + 2T\left(\frac{n}{2}\right) + C_2 \cdot n \]

\[ T(n/2) = 2T(n/4) + c \cdot \frac{n}{2} \]

Constant

divide

Combine

\[ c_1 + c_2 \]
Review of Closest-Pair Algorithm

\begin{align*}
double \ & \text{findClosestPairDist}(ptsByX, ptsByY) \{ \\
& n = ptsByX.length; \\
& \text{if} (n == 1) \text{ return } \infty; \\
& \text{if} (n == 2) \text{ return } ptsByX[0].dist(ptsByX[1]); \\
& \text{termination condition} \\
\end{align*}

\begin{align*}
& \text{divide points into left half + right half} \\
& \text{create } ptsByXLeft + ptsByYLeft \text{ of size } n/2; \\
& \text{create } ptsByXRight + ptsByYRight \text{ of size } n/2; \\
& \text{construct } ptsByXLeft + ptsByXRight \text{ from } ptsByX; \\
& \text{construct } ptsByYLeft + ptsByYRight \text{ from } ptsByY; \\
\end{align*}

\begin{align*}
& \text{conquer} \\
& d_L = \text{findClosestPairDist}(ptsByXLeft, ptsByYLeft); \\
& d_R = \text{findClosestPairDist}(ptsByXRight, ptsByYRight); \\
& d = \min(d_L, d_R); \\
& \text{create } yStrip \text{ to hold all points} \\
& \text{where } X_R - d < x-coord < X_L + d \\
& (X_R + X_L \text{ defined as in last class}) \\
& \text{as shown last class, update } d \text{ using } yStrip \\
\end{align*}

\begin{align*}
& \text{combine} \\
& \text{return } d; \\
\end{align*}
$\log_2 16 = 4$
Recursion Tree for \( T(n) = 2T(n/2) + cn \), \( T(2) = 1 \)

**Key**
- Problem size
- Time spent to split and combine for that subproblem
- Depth of recursion tree
- Leaves

\[
\log_2 n - 1 \rightarrow \frac{n}{2} \rightarrow \frac{n}{4} \rightarrow \frac{n}{8} \rightarrow \ldots \rightarrow 1
\]
\[ \text{total at level} = \begin{cases} \frac{Cn}{\log_2 n} & \text{leaves} \\ Cn & \text{levels} \end{cases} \]

\[ Cn + \frac{n}{2} \]

Asymptotic growth rate

1 statement executed per leaf
Finish time complexity analysis

\[ \sim C \cdot n \log_2 n \text{ excluding the preprocessing} \]

Sort. Use Mergesort

- divide constant \( C_1 \)
- two recursive calls on problems of size \( n/2 \)
- combine linear \( C_2 \cdot n \)

\[ T(n) = 2T(n/2) + C \cdot n \leq C' \cdot n \log_2 n \]
<table>
<thead>
<tr>
<th>Actual Time Complexity</th>
<th>( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute Force Alg</td>
<td>(~34) secs</td>
</tr>
<tr>
<td>Brute Force alg where only compute distance when ( \Delta x &lt; d + \Delta y &lt; d )</td>
<td>(~13) secs</td>
</tr>
<tr>
<td>divide + conquer</td>
<td>(~0.4) secs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>300 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
</tr>
<tr>
<td>100,000</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>( n )</td>
</tr>
<tr>
<td>( n )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Brute Force Alg</th>
<th>~34 secs</th>
<th>~77 mins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute Force alg where only compute distance when ( \Delta x &lt; d + \Delta y &lt; d )</td>
<td>~13 secs</td>
<td>~36.5 mins</td>
</tr>
<tr>
<td>divide + conquer</td>
<td>~0.4 secs</td>
<td>~17 secs</td>
</tr>
</tbody>
</table>

On 850 MHz machine, 33.7 seconds for \( n = 1,000,000 \)
When $n$ is not a power of 2

\[
\frac{n \log_2 n}{2n \log_2 (2n)} \geq 2n (\log_2 n + 1)
\]
How can we handle points that have same x-coordinate?

\( \text{ptsByXLeft} \)

\( \text{ptsByYLeft} \)

Violated that \( \text{ptsByXLeft} \) and \( \text{ptsByYLeft} \) are permutation of the same 5 points
We need a way to consistently define which points are on left and quickly determine this.

Defined left of for XYPont class defines a unique order that is sorted by x-coord.
for $i = 1$ to $n$

\[ \text{linear} \quad \{ \text{no loops, no method calls} \} \quad \text{constant time} \]

\[ \text{for } i = 1 \text{ to } n \]
\[ \quad \text{for } j = 1 \text{ to } n \]

\[ \text{for } i = 1 \text{ to } n-1 \]
\[ \quad \text{for } j = i+1 \text{ to } n \]

\[ \frac{\text{# of values}}{1} \]
\[ \frac{1}{2} \]
\[ \frac{n-1}{n(n+1)/2} \]

\[ \quad \text{quadratic} \]

\[ \quad \text{quadratic} \]
For $i = 1$ to $n-1$

For $j = 1$ to $6$

\[
\begin{align*}
\frac{i}{1} & \quad \text{# value of } j \\
\frac{1}{2} & \\
\vdots & \\
\frac{n-1}{6} & \\
\frac{6 \cdot (n-1)}{6} & \\
\end{align*}
\]

linear