5.3 Counting Inversions
Music site tries to match your song preferences with others.

- You rank n songs.
- Music site consults database to find people with similar tastes.

**Similarity metric:** number of inversions between two rankings.

- My rank: 1, 2, ..., n.
- Your rank: \(a_1, a_2, ..., a_n\).
- Songs \(i\) and \(j\) inverted if \(i < j\), but \(a_i > a_j\).

### Brute force: check all \(\Theta(n^2)\) pairs \(i\) and \(j\).
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.
### Counting Inversions: Divide-and-Conquer

**Divide-and-conquer.**

- **Divide**: separate list into two pieces.

<table>
<thead>
<tr>
<th>1</th>
<th>5</th>
<th>4</th>
<th>8</th>
<th>10</th>
<th>2</th>
<th>6</th>
<th>9</th>
<th>12</th>
<th>11</th>
<th>3</th>
<th>7</th>
</tr>
</thead>
</table>

Divide: O(1).
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- **Divide**: separate list into two pieces.
- **Conquer**: recursively count inversions in each half.

<table>
<thead>
<tr>
<th>1</th>
<th>5</th>
<th>4</th>
<th>8</th>
<th>10</th>
<th>2</th>
<th>6</th>
<th>9</th>
<th>12</th>
<th>11</th>
<th>3</th>
<th>7</th>
</tr>
</thead>
</table>

Divide: \(O(1)\).

5 blue-blue inversions 8 green-green inversions

5-4, 5-2, 4-2, 8-2, 10-2 6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7

Conquer: \(2T(n / 2)\)
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- **Divide**: separate list into two pieces.
- **Conquer**: recursively count inversions in each half.
- **Combine**: count inversions where $a_i$ and $a_j$ are in different halves, and return sum of three quantities.

Divide: $O(1)$.

Conquer: $2T(n/2)$

Combine: ???

Total = $5 + 8 + 9 = 22$. 

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>5</th>
<th>4</th>
<th>8</th>
<th>10</th>
<th>2</th>
<th>6</th>
<th>9</th>
<th>12</th>
<th>11</th>
<th>3</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5 blue-blue inversions

8 green-green inversions

9 blue-green inversions

5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7
Counting Inversions: Combine

Combine: count blue-green inversions

- Assume each half is sorted.
- Count inversions where $a_i$ and $a_j$ are in different halves.
- Merge two sorted halves into sorted whole.

13 blue-green inversions: $6 + 3 + 2 + 2 + 0 + 0$

$T(n) \leq T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n) \Rightarrow T(n) = O(n \log n)$
Counting Inversions: Implementation

Pre-condition. [Merge-and-Count] A and B are sorted.
Post-condition. [Sort-and-Count] L is sorted.

Sort-and-Count(L) {
    if list L has one element
        return 0 and the list L

    Divide the list into two halves A and B
    \( (r_A, A) \leftarrow \text{Sort-and-Count}(A) \)
    \( (r_B, B) \leftarrow \text{Sort-and-Count}(B) \)
    \( (r, L) \leftarrow \text{Merge-and-Count}(A, B) \)

    return \( r = r_A + r_B + r \) and the sorted list L
}
5.4 Closest Pair of Points
Closest Pair of Points

Closest pair. Given n points in the plane, find a pair with smallest Euclidean distance between them.

Fundamental geometric primitive.
- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, Euclidean MST, Voronoi.

Brute force. Check all pairs of points p and q with $\Theta(n^2)$ comparisons.

1-D version. $O(n \log n)$ easy if points are on a line.

Assumption. No two points have same x coordinate.

to make presentation cleaner
Closest Pair of Points: First Attempt

**Divide.** Sub-divide region into 4 quadrants.
Closest Pair of Points: First Attempt

**Divide.** Sub-divide region into 4 quadrants.

**Obstacle.** Impossible to ensure n/4 points in each piece.
Closest Pair of Points

Algorithm.

- **Divide**: draw vertical line $L$ so that roughly $\frac{1}{2}n$ points on each side.
Closest Pair of Points

Algorithm.
- **Divide:** draw vertical line L so that roughly \( \frac{1}{2}n \) 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 $\frac{1}{2}n$ points on each side.
- **Conquer:** find closest pair in each side recursively. $\leftarrow$ 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 $< \delta$. 

$\delta = \min(12, 21)$
Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < $\delta$.

- Observation: only need to consider points within $\delta$ of line L.
Closest Pair of Points

Find closest pair with one point in each side, assuming that distance $< \delta$.

- Observation: only need to consider points within $\delta$ of line $L$.
- Sort points in $2\delta$-strip by their $y$ coordinate.
Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < $\delta$.

- Observation: only need to consider points within $\delta$ of line L.
- Sort points in $2\delta$-strip by their $y$ coordinate.
- Only check distances of those within 11 positions in sorted list!

$$\delta = \min(12, 21)$$
Def. Let $s_i$ be the point in the $2\delta$-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 $\delta$.

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\(\left(p_1, \ldots, p_n\right)\) {

Compute separation line \(L\) such that half the points are on one side and half on the other side.

\[
\delta_1 = \text{Closest-Pair(}\text{left half}\text{)}
\]

\[
\delta_2 = \text{Closest-Pair(}\text{right half}\text{)}
\]

\[
\delta = \min(\delta_1, \delta_2)
\]

Delete all points further than \(\delta\) from separation line \(L\).

Sort remaining points by y-coordinate.

Scan points in y-order and compare distance between each point and next 11 neighbors. If any of these distances is less than \(\delta\), update \(\delta\).

return \(\delta\).
}

\(O(n \log n)\)

\(2T\left(\frac{n}{2}\right)\)

\(O(n)\)

\(O(n \log n)\)

\(O(n)\)
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.

\[ \Pi(n) \leq 2\Pi(n/2) + O(n) \Rightarrow \Pi(n) = O(n \log n) \]