CS Lunch
Summer Experiences
Kendade 307

Special CS Talk
Deep Networks and Doom
Bruce Maxwell
Colby College
Friday, Nov. 11, 4:00
Kendade 305

Midterm 2!
- Monday, November 21
- In class
- Covers Greedy Algorithms
- Closed book
**Divide and Conquer**

Divide-and-conquer:
- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage:
- Break up problem of size $n$ into **two equal parts of size $n/2$**.
- Solve two parts recursively.
- Combine two solutions into overall solution in **linear time**.

---

**Recommender Systems**

Netflix tries to match your movie preferences with others.
- You rank $n$ movies.
- Netflix consults database to find people with similar tastes.
- Netflix can recommend to you movies that they liked.

*Doing this well was worth $1,000,000 to Netflix!!*

---

**Counting Inversions**

Similarity metric: number of inversions between two rankings.
- My rank: $1, 2, \ldots, n$.
- Your rank: $a_1, a_2, \ldots, a_n$.
- Movies $i$ and $j$ are inverted if $i < j$, but $a_i > a_j$.

<table>
<thead>
<tr>
<th>Movies</th>
<th>Inversions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Me</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>You</td>
<td>1 3 4 2 5</td>
</tr>
</tbody>
</table>

What is the brute force algorithm?
Counting Inversions

Similarity metric: number of inversions between two rankings.
- My rank: 1, 2, ..., n.
- Your rank: a₁, a₂, ..., aₙ.
- Movies i and j are inverted if i < j, but aᵢ > aⱼ.

<table>
<thead>
<tr>
<th>Movies</th>
<th>Inversions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Me</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>You</td>
<td>1 3 4 2 5</td>
</tr>
</tbody>
</table>

What is the brute force algorithm?
Brute force: check all Θ(n²) pairs i and j.

Divide and Conquer
Count inversions relative to a sorted list

Divide into 2 sublists of equal size

divide into 2 sublists of equal size
Count inversions relative to a sorted list

Divide and Conquer

- Divide into 2 sublists of equal size
- Recursively count the inversions
- Combine by adding recursive counts and inversions across halves

Cost:

Total = 5 + 8 + 9 = 22.
Divide and Conquer

Count inversions relative to a sorted list

Divide into 2 sublists of equal size

Recursively count the inversions

Combine by adding recursive counts and inversions across halves

Cost

Total = 5 * 8 + 9 = 22.
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.
- Brute force. Check all pairs of points $p$ and $q$ with $O(n^2)$ comparisons.

Closest Pair of Points
1-dimensional version

- Sort points
- For each point, find the distance between a point and the point that follows it.
- Remember the smallest.
Closest Pair of Points

1-D version.

- Sort points
- For each point, find the distance between a point and the point that follows it.
- Remember the smallest.

Cost

$O(n \log n)$
Closest Pair of Points

1-D version.

- Sort points
- For each point, find the distance between a point and the point that follows it.
- Remember the smallest.

Total is $O(n \log n)$

Cost

$O(n \log n)$

$O(n)$

$O(n \log n)$

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.

Closest Pair of Points: First Attempt

Problem. Impossible to ensure n/4 points in each piece.

Closest Pair of Points: First Attempt

Problem. Impossible to ensure n/4 points in each piece.
Closest Pair of Points: First Attempt

Problem: Impossible to ensure n/4 points in each piece.

Closest Pair of Points

Divide: draw vertical line L so that n/2 points on each side.
Closest Pair of Points

Solve: find closest pair in each side recursively.
Closest Pair of Points

Combine: find closest pair with one point in each side.
Return best of 3 solutions.

How do we do this without comparing each point on left with each point on right???

Let $\delta$ be the minimum between pair on left and pair on right.
If there exists a pair with one point in each side and whose distance $< \delta$, find that pair.

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

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

Claim. If $|i - j| \geq 16$, then the distance between $s_i$ and $s_j$ is at least $\delta$.

Proof:
- No two points lie in same $\frac{1}{2}\delta$-by-$\frac{1}{2}\delta$ box.
- Two points separated by at least 3 rows have distance $\geq 3(\frac{1}{2}\delta)$. 

Only need to check distances of those within 15 positions in sorted list!!!!

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

Let \( s_i \) be the point in the 2δ-strip, with the \( i \)th smallest y-coordinate.

Claim. If \( |i - j| \geq 16 \), then the distance between \( s_i \) and \( s_j \) is at least \( δ \).

Proof:
- No two points lie in same \( \frac{1}{2}δ \times \frac{1}{2}δ \) box.
- Two points separated by at least 3 rows have distance \( \geq 3(\frac{1}{2}δ) \).
- If a point is within \( δ \) of point 27, it must be in one of the blue boxes.
- There are only 15 blue boxes!

Closest Pair Algorithm

Closest-Pair(p_1, …, p_n):
Compute separation line \( L \) such that half the points are on one side and half on the other side.

\( δ_1 = \text{Closest-Pair(left half)} \)
\( δ_2 = \text{Closest-Pair(right half)} \)
\( δ = \min(δ_1, δ_2) \)

Delete all points further than \( δ \) from separation line \( L \)
Sort remaining points by y-coordinate.
Scan points in y-order and compare distance between each point and next 15 neighbors. If any of these distances is less than \( δ \), update \( δ \).

return \( δ \).
Closest Pair Algorithm

Closest-Pair(p_1, ..., p_n) {
  Compute separation line L such that half the points are on one side and half on the other side.
  \( \delta_1 = \text{Closest-Pair(left half)} \)
  \( \delta_2 = \text{Closest-Pair(right half)} \)
  \( \delta = \min(\delta_1, \delta_2) \)
  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 15 neighbors. If any of these distances is less than \( \delta \), update \( \delta \).
  return \( \delta \).
}

Cost

O(n log n)

2T(n / 2)

O(n)

Cost

O(n log n)

Closest Pair Algorithm

Closest-Pair(p_1, ..., p_n) {
  Compute separation line L such that half the points are on one side and half on the other side.
  \( \delta_1 = \text{Closest-Pair(left half)} \)
  \( \delta_2 = \text{Closest-Pair(right half)} \)
  \( \delta = \min(\delta_1, \delta_2) \)
  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 15 neighbors. If any of these distances is less than \( \delta \), update \( \delta \).
  return \( \delta \).
}

Cost

O(n log n)

Closest Pair Algorithm

Closest-Pair(p_1, ..., p_n) {
  Compute separation line L such that half the points are on one side and half on the other side.
  \( \delta_1 = \text{Closest-Pair(left half)} \)
  \( \delta_2 = \text{Closest-Pair(right half)} \)
  \( \delta = \min(\delta_1, \delta_2) \)
  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 15 neighbors. If any of these distances is less than \( \delta \), update \( \delta \).
  return \( \delta \).
}

Cost

O(n log n)

2T(n / 2)

O(n)
Closest Pair Algorithm

\[ \text{Closest-Pair}(p_1, \ldots, p_n) \{ \]

- Compute separation line \( L \) such that half the points are on one side and half on the other side.
- \( \delta_1 = \text{Closest-Pair}(\text{left half}) \)
- \( \delta_2 = \text{Closest-Pair}(\text{right half}) \)
- \( \delta = \min(\delta_1, \delta_2) \)
- 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 15 neighbors. If any of these distances is less than \( \delta \), update \( \delta \).

\[ \text{return } \delta. \]

\[ \text{Cost} \]

\( O(n \log n) \)

\[ 2T(n/2) + O(n) \]

\( O(n \log n) \)

\( O(n) \)

\( 2T(n/2) + O(n \log n) \)

\( O(n \log^2 n) \)

\[ T(n) \leq 2T(n/2) + O(n \log n) \]

\[ T(n) = O(n \log^2 n) \]