Divide and Conquer

CMSC 451, Summer 2009

Group Project

• Explain project
• Choose groups and register on submit server
• Submit server test suite will be live later this week... exact submission instructions may change.

• Remember: Homework 1 due now!
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 $\frac{1}{2}n$.
- Solve two parts recursively.
- Combine two solutions into overall solution in linear time.

Consequence.
- Brute force: $n^2$.
- Divide-and-conquer: $n \log n$.

Divide et impera.
Veni, vidi, vici.
- Julius Caesar

5.1 Mergesort
Sorting

Sorting. Given n elements, rearrange in ascending order.

Applications.
- Sort a list of names.
- Organize an MP3 library.
- Display Google PageRank results.
- List RSS news items in reverse chronological order.
- Find the median.
- Find the closest pair.
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list.
- Data compression.
- Computer graphics.
- Computational biology.
- Supply chain management.
- Book recommendations on Amazon.
- Load balancing on a parallel computer.

Mergesort

Mergesort.
- Divide array into two halves.
- Recursively sort each half.
- Merge two halves to make sorted whole.

\[
\begin{array}{cccccccc}
A & L & G & O & R & I & T & H & M & S \\
A & L & G & O & R & I & T & H & M & S \\
A & G & L & O & R & H & I & M & S & T \\
A & G & H & I & L & M & O & R & S & T \\
\end{array}
\]

divide $O(1)$

sort $2T(n/2)$

merge $O(n)$
Merging

**Merging.** Combine two pre-sorted lists into a sorted whole.

**How to merge efficiently?**
- Linear number of comparisons.
- Use temporary array.

![Diagram showing merging process]

**Challenge for the bored.** In-place merge. [Kronrud, 1969]

Using only a constant amount of extra storage.

---

**Merging**

**Merge.**
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

![Diagram showing in-place merging process]
Merging

**Merge.**
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

![Diagram of merging process]

 auxiliary array
Merging

**Merge.**
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

```
smallest
A G L O R

smallest
H I M S T
```

```
AGHIL
auxiliary array
```
Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

```
A G L O R
H I M S T
```

```
A G H I L M
```

auxiliary array

Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

```
A G L O R
H I M S T
```

```
A G H I L M O
```

auxiliary array
Merging

Merge.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

Merging

Merge.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.
Merging

Merge.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

A G L O R H I M S T

A G H I L M O R S T auxiliary array

Merging

Merge.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

A G L O R H I M S T

A G H I L M O R S T auxiliary array
**A Useful Recurrence Relation**

**Def.** $T(n) =$ number of comparisons to mergesort an input of size $n$.

**Mergesort recurrence.**

$$T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + T\left(\left\lceil \frac{n}{2} \right\rceil \right) + n & \text{otherwise}
\end{cases}$$

**Solution.** $T(n) = O(n \log_2 n)$.

**Assorted proofs.** We describe several ways to prove this recurrence. Initially we assume $n$ is a power of 2 and replace $\leq$ with $=$.

---

**Proof by Recursion Tree**

$$T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
2T\left(\frac{n}{2}\right) + n & \text{otherwise}
\end{cases}$$
Proof by Telescoping

Claim. If $T(n)$ satisfies this recurrence, then $T(n) = n \log_2 n$.

\[ T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
\frac{2T(n/2)}{n} + \frac{n}{\log_2 n} & \text{otherwise}
\end{cases} \\
\text{sorting both halves} \\
\text{merging}
\]

Pf. For $n > 1$:

\[
\frac{T(n)}{n} = \frac{2T(n/2)}{n} + 1 \\
= \frac{T(n/2)}{n/2} + 1 \\
= \frac{T(n/4)}{n/4} + 1 + 1 \\
\vdots \\
= \frac{T(n/n)}{n/n} + 1 + \cdots + 1 \\
= \log_2 n \\
1
\]

Proof by Induction

Claim. If $T(n)$ satisfies this recurrence, then $T(n) = n \log_2 n$.

\[ T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
\frac{2T(n/2)}{n} + \frac{n}{\log_2 n} & \text{otherwise}
\end{cases} \\
\text{sorting both halves} \\
\text{merging}
\]

Pf. (by induction on $n$)

- Base case: $n = 1$.
- Inductive hypothesis: $T(n) = n \log_2 n$.
- Goal: show that $T(2n) = 2n \log_2 (2n)$.

\[
T(2n) = 2T(n) + 2n \\
= 2n \log_2 n + 2n \\
= 2n(\log_2 (2n) - 1) + 2n \\
= 2n \log_2 (2n) \\
1
\]

By the induction hypothesis

$\log_2 n = \log_2 (2n / 2)$
Analysis of Mergesort Recurrence

Claim. If $T(n)$ satisfies the following recurrence, then $T(n) \leq n \lfloor \lg n \rfloor$.

$$T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
T\left(\lfloor n/2 \rfloor \right) + T\left(\lceil n/2 \rceil \right) + \frac{n}{\log_2 n} & \text{otherwise}
\end{cases}$$

Pf. (by induction on $n$)
- **Base case:** $n = 1$.
- **Define** $n_1 = \lfloor n / 2 \rfloor$, $n_2 = \lceil n / 2 \rceil$.
- **Induction step:** assume true for $1, 2, \ldots, n-1$.

$$T(n) \leq T(n_1) + T(n_2) + n$$
$$\leq n_1 \lfloor \lg n_1 \rfloor + n_2 \lceil \lg n_2 \rceil + n$$
$$\leq n_1 \lfloor \lg n \rfloor + n_2 \lceil \lg n \rceil + n$$
$$= n \lfloor \lg n \rfloor + n$$
$$\leq n(\lfloor \lg n \rfloor - 1) + n$$
$$= n \lfloor \lg n \rfloor$$

$n_2 = \lceil n/2 \rceil$
$$\leq \frac{2 \lfloor \frac{n}{2} \rfloor}{2}$$
$$= 2^{\lfloor \log_2 n \rfloor} / 2$$
$$\Rightarrow \log n_2 \leq \lfloor \log n \rfloor - 1$

5.3 Counting Inversions
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: a₁, a₂, ..., aₙ.
- Songs i and j inverted if i < j, but aᵢ > aⱼ.

<table>
<thead>
<tr>
<th>Songs</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Me</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>You</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Brute force: check all Θ(n²) pairs i and j.

Applications

Applications.
- Voting theory.
- Collaborative filtering.
- Measuring the "sortedness" of an array.
- Sensitivity analysis of Google’s ranking function.
- Rank aggregation for meta-searching on the Web.
### Counting Inversions: Divide-and-Conquer

**Divide-and-conquer.**

| 1 | 5 | 4 | 8 | 10 | 2 | 6 | 9 | 12 | 11 | 3 | 7 |

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

| 1 | 5 | 4 | 8 | 10 | 2 | 6 | 9 | 12 | 11 | 3 | 7 |

Divide: $O(1)$. 

| 1 | 5 | 4 | 8 | 10 | 2 | 6 | 9 | 12 | 11 | 3 | 7 |
Counting Inversions: Divide-and-Conquer

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

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>8</td>
<td>10</td>
<td>2</td>
<td>6</td>
<td>9</td>
<td>12</td>
</tr>
</tbody>
</table>

Divide: $O(1)$.

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>8</td>
<td>10</td>
<td>2</td>
<td>6</td>
<td>9</td>
<td>12</td>
</tr>
</tbody>
</table>

Conquer: $2T(n/2)$

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

Total = 5 + 8 + 9 = 22.

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.

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>8</td>
<td>10</td>
<td>2</td>
<td>6</td>
<td>9</td>
<td>12</td>
</tr>
</tbody>
</table>

Divide: $O(1)$.

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>8</td>
<td>10</td>
<td>2</td>
<td>6</td>
<td>9</td>
<td>12</td>
</tr>
</tbody>
</table>

Conquer: $2T(n/2)$

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

Combine: ???

Total = 5 + 8 + 9 = 22.
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.

![Blue-green inversions](image)

13 blue-green inversions: 6 + 3 + 2 + 2 + 0 + 0  
Count: $O(n)$

![Merge](image)

2 3 7 10 11 14 16 17 18 19 23 25  
Merge: $O(n)$

\[
T(n) \leq T\left(\left\lceil \frac{n}{2} \right\rceil \right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + O(n) \implies T(n) = O(n \log n)
\]

Merge and Count

**Merge and count step.**

- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- **Combine** two sorted halves into sorted whole.

```
\[
i = 6
\]

```

![Merge and count](image)

```bash
Total:
```

![Auxiliary array](image)
Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.
- Combine two sorted halves into sorted whole.

\[
i = 6
\]
\[
\begin{array}{cccccccc}
3 & 7 & 10 & 14 & 18 & 19 & 2 & 11 & 16 & 17 & 23 & 25
\end{array}
\]

Total: 6
Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where \(a_i\) and \(a_j\) are in different halves.
- Combine two sorted halves into sorted whole.

\[ i = 6 \]

![](image)

Total: 6
Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.
- Combine two sorted halves into sorted whole.

\[ i = 5 \]

\[ \begin{array}{c}
3 & 7 & 10 & 14 & 18 & 19 \\
\end{array} \quad \begin{array}{c}
2 & 11 & 16 & 17 & 23 & 25 \\
\end{array} \]

\[ \text{two sorted halves} \]

\[ \begin{array}{c}
2 & 3 & 7 \\
\end{array} \]

\[ \text{auxiliary array} \]

Total: 6

---

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.
- Combine two sorted halves into sorted whole.

\[ i = 4 \]

\[ \begin{array}{c}
3 & 7 & 10 & 14 & 18 & 19 \\
\end{array} \quad \begin{array}{c}
2 & 11 & 16 & 17 & 23 & 25 \\
\end{array} \]

\[ \text{two sorted halves} \]

\[ \begin{array}{c}
2 & 3 & 7 \\
\end{array} \]

\[ \text{auxiliary array} \]

Total: 6
Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.
- Combine two sorted halves into sorted whole.

\[ i = 4 \]
\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\end{array}
\]
\[
\begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
\end{array}
\]
\[ \text{Total: 6} \]

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.
- Combine two sorted halves into sorted whole.

\[ i = 3 \]
\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\end{array}
\]
\[
\begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
\end{array}
\]
\[ \text{Total: 6} \]
Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

\[
i = 3
\]

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\hline
6 & 11 & 16 & 17 & 23 & 25 \\
\end{array}
\]

\[
\begin{array}{cccccc}
2 & 3 & 7 & 10 & 11 \\
\hline
\end{array}
\]

Total: $6 + 3$
Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.
- Combine two sorted halves into sorted whole.

\[ \begin{array}{c}
\begin{array}{ccccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\end{array} \\
\begin{array}{c}
i = 3
\end{array} \\
\end{array} \]

\[ \begin{array}{ccccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
\end{array} \]

\[ \begin{array}{c}
two \text{ sorted halves}
\end{array} \]

\[ \begin{array}{c}
2 & 3 & 7 & 10 & 11 & 14
\end{array} \]

\[ \begin{array}{c}
\text{auxiliary array}
\end{array} \]

Total: 6 + 3

Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.
- Combine two sorted halves into sorted whole.

\[ \begin{array}{c}
\begin{array}{ccccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\end{array} \\
\begin{array}{c}
i = 2
\end{array} \\
\end{array} \]

\[ \begin{array}{ccccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
\end{array} \]

\[ \begin{array}{c}
two \text{ sorted halves}
\end{array} \]

\[ \begin{array}{c}
2 & 3 & 7 & 10 & 11 & 14
\end{array} \]

\[ \begin{array}{c}
\text{auxiliary array}
\end{array} \]

Total: 6 + 3
Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

$$i = 2$$

$$\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\end{array}$$

$$\begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
\end{array}$$

two sorted halves

$$\begin{array}{cccccc}
2 & 3 & 7 & 10 & 11 & 14 & 16 \\
\end{array}$$

auxiliary array

Total: $6 + 3 + 2$
Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

\[ i = 2 \]
\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
6 & 3 & 2 & 2 \\
\end{array}
\]
\[
\begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
\end{array}
\]

Total: $6 + 3 + 2 + 2$

Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

\[ i = 2 \]
\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
6 & 3 & 2 & 2 \\
\end{array}
\]
\[
\begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
\end{array}
\]

Total: $6 + 3 + 2 + 2$
Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.
- Combine two sorted halves into sorted whole.

![Diagram of merging and counting process]

Total: 6 + 3 + 2 + 2
Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.
- Combine two sorted halves into sorted whole.

\[ \begin{array}{c}
\text{i = 1} \\
3 \ 7 \ 10 \ 14 \ 18 \ 19 \\
\text{two sorted halves}
\end{array} \]

\[ \begin{array}{c}
2 \ 3 \ 7 \ 10 \ 11 \ 14 \ 16 \ 17 \ 18 \ 19 \\
\text{auxiliary array}
\end{array} \]

Total: \( 6 + 3 + 2 + 2 \)

Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.
- Combine two sorted halves into sorted whole.

\[ \begin{array}{c}
\text{first half} \\
\text{exhausted}
\end{array} \]

\[ \begin{array}{c}
3 \ 7 \ 10 \ 14 \ 18 \ 19 \\
\text{two sorted halves}
\end{array} \]

\[ \begin{array}{c}
2 \ 3 \ 7 \ 10 \ 11 \ 14 \ 16 \ 17 \ 18 \ 19 \\
\text{auxiliary array}
\end{array} \]

Total: \( 6 + 3 + 2 + 2 \)
Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.
- Combine two sorted halves into sorted whole.

![Diagram of merging and counting process]

\[ \text{Total: } 6 + 3 + 2 + 2 + 0 \]
Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where \( a_i \) and \( a_j \) are in different halves.
- Combine two sorted halves into sorted whole.

\[
i = 0
\]

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
6 & 3 & 2 & 2 & 0 & 0
\end{array}
\begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25
\end{array}
\]

two sorted halves

\[
\begin{array}{cccccc}
2 & 3 & 7 & 10 & 11 & 14 & 18 & 19 & 23 & 25
\end{array}
\]

auxiliary array

Total: \( 6 + 3 + 2 + 2 + 0 + 0 = 13 \)
Counting Inversions: Implementation

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

```plaintext
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) ← Sort-and-Count(A)
    (r_B, B) ← Sort-and-Count(B)
    (r, L) ← 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.
  
  fast closest pair inspired fast algorithms for these problems

**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.

- **Combine**: find closest pair with one point in each side. — seems like $\Theta(n^2)$
- Return best of 3 solutions.
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.

$\delta = \min(12, 21)$
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 $L$.
- Sort points in $2\delta$-strip by their y coordinate.

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

**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 $\geq 2\left(\frac{1}{2}\delta\right)$.

Closest Pair Algorithm

```c
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 = $ Closest-Pair(left half)
    $\delta_2 = $ 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 11 neighbors. If any of these distances is less than $\delta$, update $\delta$.

    return $\delta$.
}
```

$O(n \log n)$

$2T(n / 2)$

$O(n)$

$O(n \log n)$

$O(n)$