CSE 202
Divide-and-conquer algorithms

Fan Chung Graham
UC San Diego

An induced subgraph of the collaboration graph (with Erdos number at most 2).
Made by Fan Chung Graham and Lincoln Lu in 2002.
A useful fact about trees

Any tree on $n$ vertices contains a vertex $v$ whose removal separates the remaining graph into two parts, one of which is of sizes at most $n/2$ and the other is at most $2n/3$. 
Ternary trees
A useful fact about trees

Any tree on $n$ vertices contains a vertex $v$ whose removal separates the remaining graph into two parts, one of which is of sizes at most $n/2$ and the other is at most $2n/3$.

Try to write a proof for this!
A planar graph is a graph that can be drawn in the plane without crossings.
A planar graph is a graph that can be drawn in the plane without any crossing.

Are these planar graphs?
A planar graph is a graph that can be drawn in the plane without any crossing.

Are these planar graphs?
A useful fact about planar graphs

Any planar graph on \( n \) vertices contains \( \sqrt{n} \) vertices whose removal separates the remaining graph into two parts, one of which is of sizes at most \( n/2 \) and the other is at most \( 2n/3 \).

Tarjan and Lipton, 1977
Chapter 5
Divide and Conquer
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 \textbf{two} equal parts of size \( \frac{1}{2}n \).
- Solve two parts recursively.
- Combine two solutions into overall solution in \textbf{linear time}.

Divide et impera.
Veni, vidi, vici.
- \textit{Julius Caesar}
5.1 Mergesort
Sorting

Given \( n \) elements, rearrange in ascending order.

\[
\begin{align*}
3, 6, 5, 2, 1, 4 & \quad \text{B, U, S, H} \\
1, 2, 3, 4, 5, 6 & \quad \text{B, H, S, U}
\end{align*}
\]

Obvious sorting applications.
- List files in a directory.
- Organize an MP3 library.
- List names in a phone book.
- Display Google PageRank results.
Sorting

Obvious sorting applications.
List files in a directory.
Organize an MP3 library.
List names in a phone book.
Display Google PageRank results.

Problems become easier once sorted.
Find the median.
Binary search in a database.
Identify statistical outliers.
Find duplicates in a mailing list.
Sorting

Non-obvious sorting applications.
Data compression.
Computer graphics.
Interval scheduling.
Computational biology.
Minimum spanning tree.
Supply chain management.
Simulate a system of particles.
Book recommendations on Amazon.
Load balancing on a parallel computer.
\ldots
Mergesort

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

Jon von Neumann (1945)

<table>
<thead>
<tr>
<th>A</th>
<th>L</th>
<th>G</th>
<th>O</th>
<th>R</th>
<th>I</th>
<th>T</th>
<th>H</th>
<th>M</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>L</td>
<td>G</td>
<td>O</td>
<td>R</td>
<td>I</td>
<td>T</td>
<td>H</td>
<td>M</td>
<td>S</td>
</tr>
<tr>
<td>A</td>
<td>G</td>
<td>L</td>
<td>O</td>
<td>R</td>
<td>H</td>
<td>I</td>
<td>M</td>
<td>S</td>
<td>T</td>
</tr>
<tr>
<td>A</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>R</td>
<td>S</td>
<td>T</td>
</tr>
</tbody>
</table>

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.

**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.
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.

\[
\begin{array}{c}
\text{smallest} \\
\downarrow
\end{array}
\quad
\begin{array}{c}
\text{smallest} \\
\downarrow
\end{array}
\]

\begin{array}{c}
A \\ G \\ L \\ O \\ R
\end{array}
\quad
\begin{array}{c}
H \\ I \\ M \\ S \\ T
\end{array}

\begin{array}{c}
A \\ G \\ H \\ I
\end{array}
\quad
\text{auxiliary array}
Merging

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

\[
\begin{array}{c}
\text{smallest} \\
\downarrow \\
\text{A G L O R} \\
\end{array} \quad \begin{array}{c}
\text{smallest} \\
\downarrow \\
\text{H I M S T} \\
\end{array}
\Rightarrow \begin{array}{c}
\text{A G H I L} \\
\end{array}
\text{auxiliary array}
\]
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

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.

```
smallest
A G L O R

smallest
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.

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

Auxiliary array

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

Smallest

Smallest
Merging

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

```
AGLOR

HIMST

AGHILMORS

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
Merging

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

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

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

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

(using only a constant amount of extra storage)
A Useful Recurrence Relation

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

**Mergesort recurrence.**

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

**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

\[
T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
2T(n/2) + n & \text{otherwise}
\end{cases}
\]

- **Base Case:** If \( n = 1 \), then \( T(n) = 0 \).
- **Recursive Case:** For \( n > 1 \), \( T(n) \) is the sum of the time taken to sort both halves and the time taken to merge the sorted halves.

### Diagram

- \( T(n) \) is the root node.
- \( T(n/2) \) and \( T(n/2) \) are the children nodes.
- \( T(n/4) \), \( T(n/4) \), \( T(n/4) \), and \( T(n/4) \) are the children nodes of \( T(n/2) \) and \( T(n/2) \) respectively.

The recursion tree shows that the total time \( T(n) \) is the sum of the time taken at each level, which can be expressed as:

\[
T(n) = \sum_{k=0}^{\log_2 n} \left( 2^k (n/2^k) \right) + \sum_{k=0}^{\log_2 n} \left( n \right)
\]

Simplifying this expression gives the total time as \( n \log_2 n \).
Proof by Telescoping

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

$\uparrow$

assumes $n$ is a power of 2

\[
T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
2T(n/2) + n & \text{otherwise}
\end{cases}
\]

\begin{align*}
Pf. \text{ For } n > 1: & \quad \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 \quad \text{log}_2 n \\
& = \log_2 n
\end{align*}
Proof by Induction

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

\[ T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
2T(n/2) + n & \text{otherwise}
\end{cases} \]

\[ \text{assumes } n \text{ is a power of } 2 \]

Pf. (by induction on $n$)

- **Base case:** $n = 1$.

- **Goal:** show that $T(2n) = 2n \log_2 (2n)$.

\[
T(2n) = 2T(n) + 2n \\
= 2n \log_2 n + 2n \\
= 2n(\log_2 (2n) - 1) + 2n \\
= 2n \log_2 (2n)
\]
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: *a*₁, *a*₂, ..., *a*ₙ.
- Songs *i* and *j* **inverted** if *i* < *j*, but *a*ᵢ > *a*ⱼ.

<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>

**Inversions:**
3-2, 4-2

**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.
- Nonparametric statistics (e.g., Kendall's Tau distance).
Counting Inversions: Divide-and-Conquer

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

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

\[
\begin{array}{ccccccccccc}
1 & 5 & 4 & 8 & 10 & 2 & 6 & 9 & 12 & 11 & 3 & 7 \\
\end{array}
\]

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

<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>

Conquer: $2T(n / 2)$

6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7
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.

```
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
```

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

Conquer: \( 2T(n / 2) \)

```
5 blue-blue inversions
8 green-green inversions
```

Combine: ???

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

\[
i = 6
\]

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

two sorted halves

auxiliary array

Total:
**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.

\[
i = 6
\]

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

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

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.

\[
\begin{array}{c}
i = 6 \\
3 \quad 7 \quad 10 \quad 14 \quad 18 \quad 19 \quad 2 \quad 11 \quad 16 \quad 17 \quad 23 \quad 25 \\
r \downarrow \quad \downarrow \\
\text{two sorted halves} \\
\text{auxiliary array} \\
\text{Total: 6}
\end{array}
\]
**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

3  7  10  14  18  19
   ↓
6

2  3  11  16  17  23  25
two sorted halves

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 = 5
↓
3 7 10 14 18 19
```
```
2 11 16 17 23 25
```
```
two sorted halves
```
```
6
```
```
2 3
```
```
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.

\[
i = 5
\]

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

two sorted halves

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

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.

\[ i = 4 \]
\[ 3 \quad 7 \quad 10 \quad 14 \quad 18 \quad 19 \]
\[ 2 \quad 11 \quad 16 \quad 17 \quad 23 \quad 25 \]
\[ \text{Total: 6} \]
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.

\[
\begin{array}{c}
\downarrow \\
3 & 7 & 10 & 14 & 18 & 19 \\
\downarrow \\
\end{array}
\]  \hspace{1cm}  \begin{array}{c}
\downarrow \\
2 & 11 & 16 & 17 & 23 & 25 \\
6 \\
\end{array}
\]

two sorted halves

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

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.

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\downarrow & & & & \\
3 & 7 & 10 & 14 & 18 & 19 \\
\end{array}
\quad
\begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
\downarrow & & & & \\
2 & 11 & 16 & 17 & 23 & 25 \\
\end{array}
\quad
\begin{array}{cccccc}
2 & 3 & 7 & 10 \\
\end{array}
\quad\text{two sorted halves}
\quad\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.

\[ i = 3 \]

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

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

two sorted halves

i = 3

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

auxiliary 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.

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

two sorted halves

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

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

two sorted halves

\[
\begin{array}{cccccc}
2 & 3 & 7 & 10 & 11 & 14 \\
\end{array}
\]
auxiliary 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.

\[
i = 2
\]

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

two sorted halves

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

auxiliary 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.

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

\[
\begin{array}{cccccccc}
2 & 3 & 7 & 10 & 11 & 14 & 16 & 6 & 3 & 2 \\
\end{array}
\]

\[
\text{Total: } 6 + 3 + 2
\]
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
↓
3 7 10 14 18 19
6 3 2
2 11 16 17 23 25
two sorted halves
2 3 7 10 11 14 16
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.

\[
i = 2
\]

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
6 & 3 & 2 & 2
\end{array}
\]

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

Two sorted halves

Auxiliary 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.

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

\[
i = 2
\]

\[
\begin{array}{cccc}
2 & 3 & 7 & 10 & 11 & 14 & 16 & 17 \\
\end{array}
\]

\[
\text{Total: } 6 + 3 + 2 + 2
\]
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.

\[
i = 2
\]

\[
\begin{array}{cccc}
3 & 7 & 10 & 14 \\
\hline
2 & 11 & 16 & 23 & 25
\end{array}
\]
two sorted halves

\[
\begin{array}{cccc}
2 & 3 & 7 & 10 & 11 & 14 & 16 & 17 & 18
\end{array}
\]
auxiliary 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.

\[
\begin{array}{c}
i = 1 \\
\downarrow \\
3 \quad 7 \quad 10 \quad 14 \quad 18 \quad 19 \\
\downarrow \\
2 \quad 11 \quad 16 \quad 17 \quad 23 \quad 25 \\
\end{array}
\]

\[
\begin{array}{c}
two \text{ sorted halves} \\
6 \quad 3 \quad 2 \quad 2 \\
\end{array}
\]

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

auxiliary 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.**

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

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

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.

<table>
<thead>
<tr>
<th>first half exhausted</th>
<th>i = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 7 10 14 18 19</td>
<td>2 11 16 17 23 25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>two sorted halves</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 3 2 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>auxiliary array</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 3 7 10 11 14 16 17 18 19</td>
</tr>
</tbody>
</table>

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 = 0
```

```
3 7 10 14 18 19  2 11 16 17 23 25
 6 3 2 2 0
```

```
2 3 7 10 11 14 16 17 18 19 23
```

**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.

\[
i = 0
\]

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
6 & 3 & 2 & 2 & 0 \\
\end{array}
\]

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

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.

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
6 & 3 & 2 & 2 & 0 & 0
\end{array}
\quad
\begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
i = 0
\end{array}
\quad
\begin{array}{cccccc}
2 & 3 & 7 & 10 & 11 & 14 & 16 & 17 & 18 & 19 & 23 & 25 \\
\text{two sorted halves}
\end{array}
\quad
\begin{array}{cccccc}
\text{auxiliary array}
\end{array}
\]

Total: \( 6 + 3 + 2 + 2 + 0 + 0 \)
**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.

\[
\begin{array}{c}
i = 0 \\
\downarrow \\
3 \quad 7 \quad 10 \quad 14 \quad 18 \quad 19 \quad \quad 2 \quad 11 \quad 16 \quad 17 \quad 23 \quad 25 \\
6 \quad 3 \quad 2 \quad 2 \quad 0 \quad 0
d\end{array}
\]

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

Total: \( 6 + 3 + 2 + 2 + 0 + 0 = 13 \)
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.

Count: \( O(n) \)
Merge: \( O(n) \)

13 blue-green inversions: \( 6 + 3 + 2 + 2 + 0 + 0 \)

\[
T(n) \leq T\left(\left\lfloor n/2 \right\rfloor\right) + T\left(\left\lceil n/2 \right\rceil\right) + O(n) \quad \Rightarrow \quad T(n) = O(n \log n)
\]
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 \( \frac{1}{3} n \) 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 $\frac{1}{3}n$ points on each side.
- **Conquer:** find closest pair in each side recursively.
- **Combine:** find closest pair with one point in each side. ← seems like $O(n^2)$
- 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 $L$. 

\[
\delta = \min(12, 21)
\]
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

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!

$\delta = \min(12, 21)$
**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 $\geq 2(\frac{1}{2}\delta)$.

**Fact.** Still true if we replace 12 with 7.
Closest Pair Algorithm

Closest-Pair(p₁, ..., pₙ) {
    Compute separation line L such that half the points are on one side and half on the other side.

    δ₁ = Closest-Pair(left half)
    δ₂ = Closest-Pair(right half)
    δ = min(δ₁, δ₂)

    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 11 neighbors. If any of these distances is less than δ, update δ.

    return δ.
}
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 \textit{merging} two pre-sorted lists.

\[ T(n) \leq 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n) \]
5.5 Integer Multiplication
Integer Arithmetic

Add. Given two n-digit integers $a$ and $b$, compute $a + b$.
- $O(n)$ bit operations.

Multiply. Given two n-digit integers $a$ and $b$, compute $a \times b$.
- Brute force solution: $\Theta(n^2)$ bit operations.

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
+ & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
\end{array}
\]

Add

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\times & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
\hline
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\]

Multiply
Divide-and-Conquer Multiplication: Warmup

To multiply two n-digit integers:
- Multiply four \( \frac{1}{2} n \)-digit integers.
- Add two \( \frac{1}{2} n \)-digit integers, and shift to obtain result.

\[
\begin{align*}
\frac{x}{y} &= 2^{n/2} \cdot x_1 + x_0 \\
\frac{y}{x} &= 2^{n/2} \cdot y_1 + y_0 \\
\frac{xy}{x} &= \left(2^{n/2} \cdot x_1 + x_0\right) \left(2^{n/2} \cdot y_1 + y_0\right) = 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0
\end{align*}
\]

\[
T(n) = 4T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n^2)
\]

\[
\text{assumes } n \text{ is a power of 2}
\]
Karatsuba Multiplication

To multiply two $n$-digit integers:
- Add two $\frac{1}{2}n$ digit integers.
- Multiply three $\frac{1}{2}n$-digit integers.
- Add, subtract, and shift $\frac{1}{2}n$-digit integers to obtain result.

\[
\begin{align*}
    x &= 2^{n/2} \cdot x_1 + x_0 \\
    y &= 2^{n/2} \cdot y_1 + y_0 \\
    xy &= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0 \\
        &= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_1 + x_0 y_0) - x_1 y_1 - x_0 y_0 + x_0 y_0
\end{align*}
\]

**Theorem.** [Karatsuba-Ofman, 1962] Can multiply two $n$-digit integers in $O(n^{1.585})$ bit operations.

\[
T(n) \leq T\left(\lceil n/2 \rceil \right) + T\left(\lceil n/2 \rceil \right) + T\left(1+\lceil n/2 \rceil \right) + \Theta(n)
\]

\[
\Rightarrow T(n) = O\left(n^{\log_23}\right) = O(n^{1.585})
\]
Karatsuba: Recursion Tree

\[ T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
3T(n/2) + n & \text{otherwise} 
\end{cases} \]

\[ T(n) = \sum_{k=0}^{\log_2 n} n \left(\frac{3}{2}\right)^k = \left(\frac{3}{2}\right)^{\log_2 n} - 1 = 3n^{\log_2 3} - 2 \]
Matrix Multiplication
Matrix Multiplication

Matrix multiplication. Given two n-by-n matrices $A$ and $B$, compute $C = AB$.

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

Brute force. $\Theta(n^3)$ arithmetic operations.

Fundamental question. Can we improve upon brute force?
**Matrix Multiplication: Warmup**

Divide-and-conquer.
- **Divide:** partition \( A \) and \( B \) into \( \frac{1}{2} n \)-by-\( \frac{1}{2} n \) blocks.
- **Conquer:** multiply \( 8 \frac{1}{2} n \)-by-\( \frac{1}{2} n \) recursively.
- **Combine:** add appropriate products using 4 matrix additions.

\[
\begin{bmatrix}
  C_{11} & C_{12} \\
  C_{21} & C_{22}
\end{bmatrix} = \begin{bmatrix}
  A_{11} & A_{12} \\
  A_{21} & A_{22}
\end{bmatrix} \times \begin{bmatrix}
  B_{11} & B_{12} \\
  B_{21} & B_{22}
\end{bmatrix}
\]

\[
\begin{align*}
C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\
C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\
C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\
C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22})
\end{align*}
\]

\[
T(n) = 8T(n/2) + \Theta(n^2) \quad \Rightarrow \quad T(n) = \Theta(n^3)
\]
### Matrix Multiplication: Key Idea

**Key idea.** multiply 2-by-2 block matrices with only 7 multiplications.

\[
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix} = \begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix} \times \begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
\]

\[
\begin{align*}
C_{11} &= P_5 + P_4 - P_2 + P_6 \\
C_{12} &= P_1 + P_2 \\
C_{21} &= P_3 + P_4 \\
C_{22} &= P_5 + P_1 - P_3 - P_7
\end{align*}
\]

- 7 multiplications.
- \(18 = 10 + 8\) additions (or subtractions).
Fast Matrix Multiplication

Fast matrix multiplication. (Strassen, 1969)
- **Divide:** partition A and B into $\frac{1}{2}n$-by-$\frac{1}{2}n$ blocks.
- **Compute:** 14 $\frac{1}{2}n$-by-$\frac{1}{2}n$ matrices via 10 matrix additions.
- **Conquer:** multiply 7 $\frac{1}{2}n$-by-$\frac{1}{2}n$ matrices recursively.
- **Combine:** 7 products into 4 terms using 8 matrix additions.

**Analysis.**
- Assume $n$ is a power of 2.
- $T(n) = \#$ arithmetic operations.

\[
T(n) = 7T(n/2) + \Theta(n^2) \quad \Rightarrow \quad T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})
\]
Fast Matrix Multiplication in Practice

Implementation issues.
- Sparsity.
- Caching effects.
- Numerical stability.
- Odd matrix dimensions.
- Crossover to classical algorithm around $n = 128$.

Common misperception: "Strassen is only a theoretical curiosity."
- Advanced Computation Group at Apple Computer reports 8x speedup on G4 Velocity Engine when $n \sim 2,500$.
- Range of instances where it's useful is a subject of controversy.

Remark. Can "Strassenize" Ax=b, determinant, eigenvalues, and other matrix ops.
Fast Matrix Multiplication in Theory

Q. Multiply two 2-by-2 matrices with only 7 scalar multiplications?
A. Yes! [Strassen, 1969] \[ \Theta(n^{\log_2 7}) = O(n^{2.81}) \]

Q. Multiply two 2-by-2 matrices with only 6 scalar multiplications?
A. Impossible. [Hopcroft and Kerr, 1971] \[ \Theta(n^{\log_2 6}) = O(n^{2.59}) \]

Q. Two 3-by-3 matrices with only 21 scalar multiplications?
A. Also impossible. \[ \Theta(n^{\log_3 21}) = O(n^{2.77}) \]

Q. Two 70-by-70 matrices with only 143,640 scalar multiplications?
A. Yes! [Pan, 1980] \[ \Theta(n^{\log_{70} 143640}) = O(n^{2.80}) \]

Decimal wars.
- December, 1979: \( O(n^{2.521813}) \).
- January, 1980: \( O(n^{2.521801}) \).
Fast Matrix Multiplication in Theory

**Best known.** $O(n^{2.376})$ [Coppersmith-Winograd, 1987.]

**Conjecture.** $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$.

**Caveat.** Theoretical improvements to Strassen are progressively less practical.