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.
Announcements

• Homework due today before the class.
• About homework, write your own homework, allowing oral discussion with one fixed partner.
• Fan’s office hour will be held at CSE2126 this week.
• Olivia’s office hour will be changed.
Chapter 4

Greedy Algorithms
4.4 Shortest Paths in a Graph

Shortest path tree in Bay area
Greed is good. Greed is right. Greed works. Greed clarifies, cuts through, and captures the essence of the evolutionary spirit.

- Gordon Gecko (Michael Douglas)
Trees with at most 4 edges
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 \( 2n/3 \).
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 \( c\sqrt{n} \) 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

\[ c = 2\sqrt{2} \]
Chapter 5
Divide and Conquer
Sorting

Given n elements, rearrange in ascending order.

3, 6, 5, 2, 1, 4  B, U, S, H
1, 2, 3, 4, 5, 6  B, H, S, U

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

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

Jon von Neumann (1945)

| ALGORSITHTMS | ALGORSITHTMS | AGLORHMST | AGHILMORS | 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.
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

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
```

auxiliary array
Merging

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

\[
\begin{align*}
  \text{smallest} & \quad \text{smallest} \\
  \begin{array}{cccc}
    A & G & L & O \\
  \end{array} & \quad \begin{array}{cccc}
    H & I & M & S \\
  \end{array} \\
  \begin{array}{cccc}
    A & G & H & I \\
  \end{array} & \quad \text{auxiliary array}
\end{align*}
\]
Merging

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

```plaintext
AGLOR
```
```
HIMS
```
```
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.

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

auxiliary array
**Merging**

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

![Diagram showing merging of two sorted arrays with auxiliary array](attachment:image.png)
Merging

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

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

auxiliary array
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. Combine two pre-sorted lists into a sorted whole.

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

![Diagram showing merging of two lists]

- List A: A, G, H, I
- List B: R, E, M, S, T

In-place merge (Kronrud, 1969)
A Useful Recurrence Relation

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

Mergesort recurrence.

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

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

Proof: First try the recurrence \( T(n) \leq 2T(n/2) + n \)

Prove by induction: Suppose for \( k < n, T(k) \leq ck \log k \).

\[
T(n) \leq 2c(n/2)\log(n/2) + n \\
\leq cn(\log n - 1) + n \\
\leq cn \log n
\]
Solving recurrences:

\[ T(n) = 2T(n/2) + n \quad \Rightarrow \quad T(n) = O(n \log n) \]
Solving recurrences:

\[ T(n) = 2T(n/2) + n \quad \Rightarrow \quad T(n) = O(n \log n) \]

\[ T(n) = 2T(n/2) + n^2 \]
Solving recurrences:

\[ T(n) = 2T(n/2) + n \quad \Rightarrow \quad T(n) = O(n \log n) \]

\[ T(n) = 2T(n/2) + n^2 \quad \Rightarrow \ ? \]
Solving recurrences:

\[ T(n) = 2T\left(\frac{n}{2}\right) + n \quad \Rightarrow \quad T(n) = O(n \log n) \]

\[ T(n) = 2T\left(\frac{n}{2}\right) + n^2 \quad \Rightarrow \ ? \]

\[ T(n) = 3T\left(\frac{n}{2}\right) + n \log n \]
Solving recurrences:

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

\[ T(n) = 2T(n/2) + n^2 \implies ? \]

\[ T(n) = 3T(n/2) + n\log n \implies ? \]
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.

Inversions: 3-2, 4-2
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.

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

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

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:** $O(1)$.

**Conquer:** $2T(n/2)$

- 5 blue-blue inversions: $5-4, 5-2, 4-2, 8-2, 10-2$
- 8 green-green inversions: $6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7$
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.

<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>
</table>

Divide: $O(1)$.

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.
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.

\[ i = 6 \]

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

two sorted halves

auxiliary array

Total:
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.

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

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.

<table>
<thead>
<tr>
<th>$i = 6$</th>
<th>3</th>
<th>7</th>
<th>10</th>
<th>14</th>
<th>18</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2</th>
<th>11</th>
<th>16</th>
<th>17</th>
<th>23</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Two sorted halves: 6

Auxiliary array:

```
2
```

**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.

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

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.

\[
\begin{array}{ccccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
i = 5
\end{array}
\quad
\begin{array}{ccccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
\downarrow
\end{array}
\]

Two sorted halves

\[
\begin{array}{cccc}
2 & 3
\end{array}
\]

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.*

---

\[ i = 5 \]

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

\[
\begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
6 & & & & & \\
\end{array}
\]

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.

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

\[ i = 4 \]

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.

```
merge: 3  7  10  14  18  19
        ↓
       count: 2  3  7  10
         auxiliary array

merge: 2  11  16  17  23  25
        ↓
       count: 6
    two sorted halves

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.

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

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

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.

<table>
<thead>
<tr>
<th>3</th>
<th>7</th>
<th>10</th>
<th>14</th>
<th>18</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>11</td>
<td>16</td>
<td>17</td>
<td>23</td>
<td>25</td>
</tr>
</tbody>
</table>

$$i = 3$$

Two sorted halves

| 2 | 3 | 7 | 10 | 11 |

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.

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

\[
\begin{array}{cccccccc}
6 & \quad & \quad & \quad & \quad & \quad \\
3 & \quad & \quad & \quad & \quad & \quad \\
\end{array}
\]

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

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.

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

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

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.

---

<table>
<thead>
<tr>
<th>3</th>
<th>7</th>
<th>10</th>
<th>14</th>
<th>18</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>11</td>
<td>16</td>
<td>17</td>
<td>23</td>
<td>25</td>
</tr>
</tbody>
</table>

$i = 2$

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.

$$i = 2$$

$3 \ 7 \ 10 \ 14 \ 18 \ 19$

$2 \ 11 \ 16 \ 17 \ 23 \ 25$

Two sorted halves

6
3
2

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.

\[
i = 2
\]

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

\[
\begin{array}{cccccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
6 & 3 & 2 \\
\end{array}
\]

two sorted halves

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

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.

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

\[ \begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
6 & 3 & 2 & 2 \\
\end{array} \]

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>i = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 7 10 14 18 19</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</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.

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

two sorted halves

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

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

\[
\begin{align*}
& \quad \text{i = 1} \\
& \quad \downarrow \\
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\end{array} & \quad \downarrow \\
\begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
6 & 3 & 2 & 2 \\
\end{array} & \quad \text{two sorted halves} \\
\begin{array}{cccccc}
2 & 3 & 7 & 10 & 11 & 14 & \text{auxiliary array} \\
16 & 17 & 18 \\
\end{array} & \quad \text{auxiliary array} \\
\end{align*}
\]

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.

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

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.

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

**first half exhausted**  $i = 0$

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

**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.

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

\[
\text{Total: } 6 + 3 + 2 + 2 + 0
\]
**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.

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

\[
\begin{align*}
&\quad \\
i = 0 & \quad \\
3 & \quad 7 & \quad 10 & \quad 14 & \quad 18 & \quad 19 & \quad 2 & \quad 11 & \quad 16 & \quad 17 & \quad 23 & \quad 25 \\
6 & \quad 3 & \quad 2 & \quad 2 & \quad 0 & \quad 0
\end{align*}
\]

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.

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

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

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.

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

Count: \( O(n) \)

Merge: \( O(n) \)

\[
T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n) \quad \Rightarrow \quad T(n) = O(n \log n)
\]
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) ← 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
}
Merge-and-Count($A,B$) {
  Initialize $\text{Pointer1}$ to the front of $A$.
  $\text{Pointer2}$ to the front of $B$.
  $\text{Count} = 0$

  While $A$ and $B$ are nonempty,

  compare $a_i$ at $\text{Pointer1}$ with $b_j$ at $\text{Pointer2}$,
  append the smaller one to output and advance the pointer by one.

  If $b_j$ is smaller, then increase $\text{Count}$ by the number of elements still in $A$.

  Endwhile}
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.

\[ \text{fast closest pair inspired fast algorithms for these problems} \]

\[ \text{to make presentation cleaner} \]
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.
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.
- **Combine:** find closest pair with one point in each side. $\searrow$ 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 $< \delta$. 

\[ \delta = \min(12, 21) \]
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.
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!
**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 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.
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.

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

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

assumes \( n \) 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.

\[
\begin{align*}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= 2^n \cdot x_1y_1 + 2^{n/2} \cdot (x_1y_0 + x_0y_1) + x_0y_0 \\
&= 2^n \cdot x_1y_1 + 2^{n/2} \cdot \left( (x_1 + x_0)(y_1 + y_0) - x_1y_1 - x_0y_0 \right) + x_0y_0
\end{align*}
\]

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

\[
\begin{align*}
T(n) &\leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1+\lceil n/2 \rceil) + \Theta(n) \\
&\text{recursive calls add, subtract, shift} \\
\Rightarrow T(n) &= O(n^{\log_2 3}) = O(n^{1.585})
\end{align*}
\]
Karatsuba: Recursion Tree

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

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

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

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

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.

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

\[
C_{11} = (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\
C_{12} = (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\
C_{21} = (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\
C_{22} = (A_{21} \times B_{12}) + (A_{22} \times B_{22})
\]

\[
T(n) = 8T(n/2) + \Theta(n^2) \quad \Rightarrow \quad T(n) = \Theta(n^3)
\]

- recursive calls
- add, form submatrices
Matrix Multiplication: Key Idea

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

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

- \( P_1 = A_{11} \times (B_{12} - B_{22}) \)
- \( P_2 = (A_{11} + A_{12}) \times B_{22} \)
- \( P_3 = (A_{21} + A_{22}) \times B_{11} \)
- \( P_4 = A_{22} \times (B_{21} - B_{11}) \)
- \( P_5 = (A_{11} + A_{22}) \times (B_{11} + B_{22}) \)
- \( P_6 = (A_{12} - A_{22}) \times (B_{21} + B_{22}) \)
- \( P_7 = (A_{11} - A_{21}) \times (B_{11} + B_{12}) \)

- 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.
- $T(n) = \#\text{ arithmetic operations}.$

\[
T(n) = 7T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})
\]
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 ~ 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.** $O(n^{2+\epsilon})$ for any $\epsilon > 0$.

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