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

- 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 \( \frac{n}{2} \).
- Solve two parts recursively.
- Combine two solutions into overall solution in linear time.

Consequence.
- \( \text{Brute Force: } n^2 \).
- \( \text{Divide-and-conquer: } n \log n \).

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

5.1 Mergesort

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

\[
\text{merge sort} \quad \frac{A}{L} \quad \frac{G}{O} \quad \frac{R}{I} \quad \frac{M}{S} \\
\frac{S}{A} \quad \frac{G}{L} \quad \frac{O}{R} \quad \frac{H}{I} \quad \frac{M}{S} \\
\quad \frac{T}{A} \quad \frac{G}{L} \quad \frac{O}{R} \quad \frac{H}{I} \quad \frac{M}{S} \\
\quad \frac{O(n)}{2T(n/2)} \quad \frac{O(1)}{merge \ O(n)}
\]

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.

5.1 Mergesort

Applications.
- Divide array into two halves.
- Recursively sort each half.
- Merge two halves to make sorted whole.
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. [Knuth, 1969]

using only a constant amount of extra storage

Merging

Merging.

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

Merging

Merging.

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

Merging

Merging.

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

Merging

Merging.

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

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

### Auxiliary Array

- smallest
- A G L O R
- H I M S T

### Auxiliary Array

- smallest
- A G H I L M O

### Auxiliary Array

- smallest
- A G H I L M O R

### Auxiliary Array

- smallest
- A G H I L M O R S

### Auxiliary Array

- smallest
- A G H I L M O R S T

### Auxiliary Array

- first half exhausted
- smallest
- A G L O R
- H I M S T

### Auxiliary Array

- first half exhausted
- second half exhausted
- A G L O R
- H I M S T

### Auxiliary Array

- first half exhausted
- second half exhausted
- A G H I L M O R S T
Initially we assume $n$ is a power of 2 and replace $n$ with $n/2$.

**Proof by Recursion Tree**

Claim: If $T(n)$ satisfies this recurrence, then $T(n) = \Theta(n \log n)$.

Proof by Recursion Tree

Claim: If $T(n)$ satisfies this recurrence, then $T(n) = \Theta(n \log n)$.

Proof by Induction

Claim: If $T(n)$ satisfies this recurrence, then $T(n) = \Theta(n \log n)$.

Proof by Induction

**5.3 Counting Inversions**
Music site tries to match your song preferences with others.
- You rank $n$ songs.
- Music site consults database to find people with similar tastes.

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

<table>
<thead>
<tr>
<th>Song</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 $\Theta(n^2)$ pairs $i$ and $j$.

---

### 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>1 5 4 8 10 2 6 9 12 11 3 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Divide: $O(1)$</td>
</tr>
<tr>
<td>Conquer: $2T(n/2)$</td>
</tr>
<tr>
<td>Combine: ??</td>
</tr>
</tbody>
</table>

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

---

### 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: 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

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: $O(n)$

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.

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.

\[ i = 5 \]

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

**two sorted halves**

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

**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.

\[ i = 4 \]

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

**two sorted halves**

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

**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.

\[ i = 3 \]

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

**two sorted halves**

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

**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.

\[ i = 3 \]

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

**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.

```
i = 3
```

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>16</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
i = 2
```

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>16</td>
<td>23</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Total: $6 + 3$

```
i = 2
```

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>16</td>
<td>23</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
i = 2
```

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>16</td>
<td>23</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Total: $6 + 3 + 2$

```
i = 2
```

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>16</td>
<td>23</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
i = 2
```

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>16</td>
<td>23</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Total: $6 + 3 + 2 + 2$

```
i = 2
```

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>16</td>
<td>23</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
i = 2
```

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>16</td>
<td>23</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Total: $6 + 3 + 2 + 2$

```
i = 2
```

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>16</td>
<td>23</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
i = 2
```

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>16</td>
<td>23</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Total: $6 + 3 + 2 + 2$

```
i = 2
```

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>16</td>
<td>23</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
i = 2
```

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>16</td>
<td>23</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></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.

\[
\begin{align*}
\text{Total: } 6 + 3 + 2 + 2
\end{align*}
\]
Merge and Count

Merge 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.

\[
i = 0
\]

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

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. 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 inspires 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.
Obstacle: Impossible to ensure \( \frac{n}{4} \) points in each piece.

Closest Pair of Points

Algorithm:
- Divide: draw vertical line \( L \) so that roughly \( \frac{n}{2} \) points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side. Return best of 3 solutions.

Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < \( h \).
Observation: only need to consider points within \( h \) of line \( L \).

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

Find closest pair with one point in each side, assuming that distance < δ.

- Observation: only need to consider points within δ of line L.
- Sort points in 2δ-strip by their y coordinate.

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

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.
    \( \delta_1 = \text{Closest-Pair(left half)} \)
    \( \delta_2 = \text{Closest-Pair(right half)} \)
    \( \delta = \min(\delta_1, \delta_2) \)
    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 11 neighbors. If any of these distances is less than \( \delta \), update \( \delta \).
    return \( \delta \).
}
```

\( O(n \log n) \)

\( 2T(n / 2) \)

\( O(n) \)

\( O(n \log \log n) \)

\( O(n) \)