Chapter 5
Divide and Conquer

5.1 Mergesort

Mergesort

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

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.

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

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

Jon von Neumann (1945)

$O(n)$

$2T(n/2)$

$\Theta(n)$

$\Theta(n \log n)$
**Merging**

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

How to merge efficiently?

- Linear number of comparisons.
- Use temporary array.

```
AGLOR

HIMST

AGHI
```

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.

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

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)

2T(n/2) + \alpha

T(n/4)

T(n/4)

T(n/4)

T(n/4)

T(2)

T(2)

T(2)

T(2)

T(2)

T(2)
```

**Proof by Telescoping**

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

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

Pf. For $n > 1$:

$$T(n) = \begin{cases} 
2T(n/2) + \alpha & \text{if } n = 1 \\
2T(n/2) + \alpha & \text{otherwise}
\end{cases}$$

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

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

<table>
<thead>
<tr>
<th>Inversions</th>
<th>3-2, 4-2</th>
</tr>
</thead>
</table>

Brute force: check all Θ(n²) pairs i and j.
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

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

Divide: \( O(1) \).

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

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
```

Total = 5 + 8 + 9 = 22.
5.4 Closest Pair of Points

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.

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

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

Count: $O(n)$

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

Merge: $O(n)$

```
T(n) ≤ T(\lceil n/2 \rceil) + T(\lceil n/2 \rceil) + O(n) \Rightarrow 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
    \((r_A, A) \leftarrow \text{Sort-and-Count}(A)\)
    \((r_B, B) \leftarrow \text{Sort-and-Count}(B)\)
    \((r_A, L) \leftarrow \text{Merge-and-Count}(A, B)\)
    return \(r = r_A + r_B + r\) and the sorted list L
}
```

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.

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

Closest Pair of Points

Algorithm.
- Divide: draw vertical line L so that roughly 0.5n 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. — 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 \( L \).
- 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!

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

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 \( \delta \).

\[ \text{Pf.} \]

- 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

```plaintext
Closest-Pair(p_1, ..., p_n) {
    Compute separation line \( L \) such that half the points are on one side and half on the other side.
    \( \delta_1 = \text{Closest-Pair(left half)} \)
    \( \delta_2 = \text{Closest-Pair(right half)} \)
    \( \delta = \min(\delta_1, \delta_2) \)
    Delete all points further than \( \delta \) from separation line \( L \)
    Sort remaining points by y-coordinate.
    Scan points in y-order and compare distance between each point and next 11 neighbors. If any of these distances is less than \( \delta \), update \( \delta \).
    return \( \delta \).
}
```

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.