Divide and Conquer

Divide-and-Conquer

- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions of 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 (sorting):
- Brute force: $n^2$
- Divide-and-conquer: $n \log n$

5.3 Counting Inversions

Counting Inversions: Divide-and-Conquer

Divide-and-Conquer

- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions of sub-problems into overall solution.

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: $1, 2, \ldots, n$.
- Your rank: $a_1, a_2, \ldots, a_n$.
- Songs $i$ and $j$ inverted if $i < j$ but $a_i > a_j$.

Inversions

<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>My</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Your</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Brute force: check all $\binom{n}{2}$ pairs $i$ and $j$.

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.

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

Total: $5 + 8 + 9 = 22$

Int: $T(n) \leq T(n/2) + T(\lceil n/2 \rceil) + O(n)$

$T(n) = O(n \log n)$
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.

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.

\[ L \]

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 \( n/2 \) 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.
- 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.
- Only check distances of those within 11 positions in sorted list!

Def. Let $s_i$ be the point in the $2\delta$-strip, with the $i$th smallest $y$-coordinate.

Claim. If $|i - j| \geq 11$, 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 $\geq 2(\frac{1}{2}\delta)$. □
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 \( \delta \).
}

Closest Pair of Points: Analysis

Running time.
\[ T(n) \leq 2T(n/2) + O(n \log n) \]
\[ T(n) = O(n \log^2 n) \]

Complex Multiplication

Complex multiplication. \((a + bi)(c + di) = x + yi\).

Grade-school. \( x = ac - bd, y = bc + ad \).

Q. Is it possible to do with fewer multiplications?
A. Yes. [Gauss] \( x = ac - bd, y = bc + ad, x = (a+b)(c+d) - ac - bd \).

Remark. Improvement if no hardware multiply.

5.5 Integer Multiplication

Addition. Given two \( n \)-bit integers \( a \) and \( b \), compute \( a + b \).

Grade-school. \( n \) \( n \)-bit operations.

Remark. Grade-school addition algorithm is optimal.
Divide-and-Conquer Multiplication: Warmup

To multiply two $n$-bit integers $a$ and $b$:

- Multiply four $\frac{n}{2}$-bit integers, recursively.
- Add and shift to obtain result.

\[
\begin{align*}
T(2) &= \begin{cases} 
1 & \text{if } n = 1 \\
2 & \text{if } n > 1 
\end{cases} \\
T(n) &= T(n/2) + T(n/4) + \frac{3n}{2} 
\end{align*}
\]

Example:

\[
\begin{align*}
a &= 10001001 \\
b &= 11100101 \\
a \cdot b &= 11100101 \\
\end{align*}
\]

Theorem. [Karatsuba-Ofman 1962] Can multiply two $n$-bit integers in $O(n^{\log_2 3})$ bit operations:

\[
T(n) = 3 \cdot T(n/2) + \frac{3n}{2} \quad \text{if } n \geq 2 \\
T(1) = 1 
\]
Matrix Multiplication

Dot Product
Given two length-
length vectors \(a\) and \(b\), compute \(c = a \cdot b\).

Grade-school: \(n^2\) arithmetic operations.

\[
a = \begin{bmatrix} 70 & 20 & 30 \\ 30 & 40 & 30 \end{bmatrix}
\]
\[
b = \begin{bmatrix} 30 \\ 40 \\ 30 \end{bmatrix}
\]
\[
a \cdot b = (70 \times 30) + (20 \times 40) + (30 \times 30) = 32
\]

Remark. Grade-school dot product algorithm is optimal.

Matrix Multiplication
Given two \(n \times n\) matrices \(A\) and \(B\), compute \(C = AB\).

Grade-school: \(n^3\) arithmetic operations.

\[
C = AB
\]

Remark. Grade-school matrix multiplication algorithm is optimal.

Block Matrix Multiplication

To multiply two \((n \times n)\) matrices \(A\) and \(B\):

- **Divide**: partition \(A\) into \(\frac{n}{2} \times \frac{n}{2}\) blocks.
- **Conquer**: multiply \(\frac{n}{2} \times \frac{n}{2}\) blocks recursively.
- **Combine**: add appropriate products using \(n^2\) matrix additions.

\[
C_n = A_n \cdot B_n + A_n \cdot B_n
\]

Matrix Multiplication: Warmup

To multiply two \(n \times n\) matrices \(A\) and \(B\):

- **Divide**: partition \(A\) into \(\frac{n}{2} \times \frac{n}{2}\) blocks.
- **Conquer**: multiply \(\frac{n}{2} \times \frac{n}{2}\) blocks recursively.
- **Combine**: add appropriate products using \(n^2\) matrix additions.

\[
C_n = C_{n/2} \cdot C_{n/2} + C_{n/2} \cdot C_{n/2}
\]

Fast Matrix Multiplication

Key idea. Multiply \(2 \times 2\) blocks with only \(7\) multiplications.

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

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

To multiply two \( n \times n \) matrices \( A \) and \( B \): 

- Divide: partition \( A \) and \( B \) into \( \frac{1}{2}n \times \frac{1}{2}n \) blocks.
- Compute: 14 \( \frac{1}{2}n \times \frac{1}{2}n \) matrices via 10 matrix additions.
- Conquer: multiply 7 pairs of \( \frac{1}{2}n \times \frac{1}{2}n \) matrices, recursively.
- Combine: 7 products into 4 terms using 8 matrix additions.

Analysis.

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

\[
T(n) = 7T(\frac{n}{2}) + O(n^2) \\
T(n) = O(n^{\log_2 7}) = O(n^{2.81})
\]

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.”

- Apple reports 8x speedup on G4 Velocity Engine when \( n \approx 2,500 \).
- Range of instances where it’s useful is a subject of controversy.

Remark. Can “Strassenize” \( Ax = b \), determinant, eigenvalues, SVD, ...

Fast Matrix Multiplication: Theory

Q. Multiply two 2-by-2 matrices with 7 scalar multiplications?
   A. Yes!  [Strassen 1969]

Q. Multiply two 2-by-2 matrices with 6 scalar multiplications?
   A. Impossible. [Hopcroft and Kerr 1971]

Q. Two 3-by-3 matrices with 21 scalar multiplications?
   A. Also impossible.

\[
T(20) = O(n^{2.805}) \\
T(48) = O(n^{2.521813})
\]

Relevant works: [Pan, Bini et al., Schönhage, ...]

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

Conjecture. \( O(n^{\alpha}) \) for any \( \alpha > 0 \).

Conject. Theoretical improvements to Strassen are progressively less practical.