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 \( \frac{n}{2} \) 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

5.1 Mergesort

Sorting. Given n elements, rearrange in ascending order.

Obvious sorting applications.
List files in a directory.
Organize a playlist.
List names in address book.
Display Google PageRank results.

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

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.
...
Mergesort

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

A Useful Recurrence Relation

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

Mergesort recurrence.

$$T(n) = \begin{cases} 
0 & \text{if } n=1 \\
T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + \frac{n}{\log_2 n} & \text{otherwise}
\end{cases}$$

Solution. $T(n) = \Theta(n \log_2 n)$.

Proof by Telescoping

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

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

Pf. For $n > 1$:

$$\frac{T(n)}{n} = \frac{2T(n/2)}{n} + \frac{1}{\log_2 n}$$

$$= \frac{T(n/4)}{n/2} + \frac{1}{\log_2 n}$$

$$= \frac{T(n/8)}{n/4} + \frac{1}{\log_2 n}$$

$$\cdots$$

$$= \frac{T(n/2^n)}{n/2^n} + \frac{1}{\log_2 n}$$

$$= \frac{n}{n} + \frac{1}{\log_2 n}$$

$$= \log_2 n$$
Proof by Induction

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

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

Pf. (by induction on \( n \))
- Base case: \( n = 1 \).
- Inductive hypothesis: \( T(n) = n \log_2 n \).
- Goal: show that \( T(2n) = 2n \log_2 (2n) \).

\[
T(2n) = 2T(n) + 2n \\
= 2n \log_2 2 + 2n \\
= 2n(\log_2 (2n) + 1) + 2n \\
= 2n \log_2 (2n)
\]

Analysis of Mergesort Recurrence

Claim. If \( T(n) \) satisfies the following recurrence, then \( T(n) \leq n \lfloor \lg n \rfloor \).

\[
T(n) \leq \begin{cases} 
0 & \text{if } n = 1 \\
T(\lfloor n/2 \rfloor) + 7(\lceil n/2 \rceil) + n & \text{otherwise}
\end{cases}
\]

Pf. (by induction on \( n \))
- Base case: \( n = 1 \).
- Inductive hypothesis: \( T(n) \leq n \lfloor \lg n \rfloor \).
- Induction step: assume true for \( 1, 2, \ldots, n-1 \).

\[
T(n) \leq T(n_1) + T(n_2) + n \\
\leq n_1 \lfloor \lg n_1 \rfloor + n_2 \lfloor \lg n_2 \rfloor + n \\
\leq n \lfloor \lg n \rfloor + n \\
\leq n (\lfloor \lg n \rfloor - 1) + n \\
= n \lfloor \lg n \rfloor
\]

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 \( a_i > a_j \).

**Songs**

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>You</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Inversions
- 3-2, 4-2

Brute force: check all \( \Theta(n^2) \) pairs \( i \) and \( j \).
Applications

- Voting theory.
- Collaborative filtering.
- Measuring the "sortedness" of an array.
- Genomic distance between two gene sequences.
- 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.

- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half.

Divide: O(1).

Conquer: 2T(n / 2)

5 blue-blue inversions
8 green-green inversions

5-4, 5-2, 4-2, 8-2, 10-2
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.

<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**: \( O(1) \)

<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**: \( 2T(n/2) \)

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

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: \( O(n) \)

5.4 Closest Pair of Points

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
}
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.

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.

Find closest pair with one point in each side, assuming that distance < \( \delta \).

- Observation: only need to consider points within \( \delta \) of line \( L \).
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

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(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 \( \delta \).
}

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

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

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.

\[ x = 2^n x_1 + x_0 \]
\[ y = 2^n y_1 + y_0 \]
\[ xy = (2^n x_1 + x_0)(2^n y_1 + y_0) = 2^n x_1 y_1 + 2^n x_1 y_0 + x_0 y_1 + x_0 y_0 \]

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

assumed \( n \) 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.

\[
\begin{align*}
x &= 2^n x_1 + x_0 \\
y &= 2^n y_1 + y_0 \\
x'y &= 2^n x_1 y_1 + 2^{n-1} (x_1 y_0 + x_0 y_1) + x_0 y_0
\end{align*}
\]

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

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

\[
T(n) = \sum_{k=0}^{\log_2 n} \left( \left\lceil \frac{n}{2^k} \right\rceil \right)^3 = \frac{\left( \frac{n}{2} \right)^3 - 1}{3} = \frac{3n^3}{8} - \frac{3n}{2} + \frac{1}{4}
\]

Karatsuba: Recursion Tree

\[
\begin{align*}
T(\infty) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n) \\
\Rightarrow T(n) &= O(4^{\log_2 3}) = O(n^{1.585})
\end{align*}
\]

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-$\frac{1}{2}n$ matrices recursively.
- **Combine:** add appropriate products using 4 matrix additions.

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

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

\[
T(n) = 8T(n/2) + \Theta(n^3) \Rightarrow T(n) = \Theta(n^{\log_2 8}) = \Theta(n^{2.81})
\]

Matrix Multiplication: Key Idea

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

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

- $7$ 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.

\[
T(n) = 7T(n/2) + \Theta(n^3) \Rightarrow T(n) = \Theta(n^{\log_2 7}) = \Theta(n^{2.81})
\]

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

Best known. \( O(n^{2.3728639}) \) [Francois Le Gall, 2014]

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

Caveat. Theoretical improvements to Strassen are progressively less practical. The constant coefficient hidden by the Big O notation is so large that these algorithms are only worthwhile for matrices that are too large to handle on present-day computers.