5. Divide-and-Conquer

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

5.1 Mergesort

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 $\frac{1}{2}n$.
- Solve two parts recursively.
- Combine two solutions into overall solution in linear time.

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

Sorting

Given $n$ elements, rearrange in ascending order.

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

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.

Problems become easier once sorted.
- Find the median.
- Find the closest pair.
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list.
A Useful Recurrence Relation

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

Mergesort recurrence.

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

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 \( \approx \) with \( = \).

Proof by Recursion Tree

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. [Kronrud, 1969]

using only a constant amount of extra storage

Mergesort

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

Jon von Neumann (1945)
Claim. If $T(n)$ satisfies this recurrence, then $T(n) = n \log_2 n$.

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

\[
T(2n) = 2T(n) + 2n = 2n \log_2 n + 2n = 2n \log_2 (2n) - 1 + 2n = 2n \log_2 (2n)
\]
**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 \( a_i > a_j \).

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

<table>
<thead>
<tr>
<th>Songs</th>
<th>Inversions</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D E</td>
<td>3-2, 4-2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Me</th>
<th>1 2 3 4 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>You</td>
<td>1 3 4 2 5</td>
</tr>
</tbody>
</table>

**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.

\[ 1 \ 5 \ 4 \ 8 \ 10 \ 2 \ 6 \ 9 \ 12 \ 11 \ 3 \ 7 \]

Divide:
- **Divide**: separate list into two pieces.

\[ 1 \ 5 \ 4 \ 8 \ 10 \ 2 \ 6 \ 9 \ 12 \ 11 \ 3 \ 7 \]

\[ 1 \ 5 \ 4 \ 8 \ 10 \ 2 \ 6 \ 9 \ 12 \ 11 \ 3 \ 7 \]
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.

\[
\begin{align*}
\text{Divide: } & O(1) \\
\text{Conquer: } & 2T(n/2) \\
\text{Combine: } & \text{count} \end{align*}
\]

\[
\begin{align*}
\text{Divide: } & O(1) \\
\text{Conquer: } & 2T(n/2) \\
\text{Combine: } & 3 \text{ quantities}
\end{align*}
\]

Counting Inversions: Implementation

**Pre-condition.** [Merge-and-Count] A and B are sorted.

**Post-condition.** [Sort-and-Count] L is sorted.

```plaintext
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
}
```

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.

\[
\begin{align*}
\text{Divide: } & O(1) \\
\text{Conquer: } & 2T(n/2) \\
\text{Combine: } & \text{count} \end{align*}
\]

\[
\begin{align*}
\text{Divide: } & O(1) \\
\text{Conquer: } & 2T(n/2) \\
\text{Combine: } & 3 \text{ quantities}
\end{align*}
\]

\[
\begin{align*}
13 \text{ blue-green inversions: } & 6 + 3 + 2 + 0 + 0 \\
\text{Count: } & O(n) \\
2 + 3 + 7 + 10 + 11 + 14 + 16 + 17 + 18 + 19 + 23 + 25 \text{ Merge: } & O(n)
\end{align*}
\]

\[
T(n) \leq T\left(\left\lceil n/2 \right\rceil\right) + T\left(\left\lfloor n/2 \right\rfloor\right) + O(n) \quad \Rightarrow \quad T(n) = O(n \log n)
\]
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, 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 $\frac{1}{2}n$ points on each side.

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

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. $\sim$ seems like $O(n^{1.5})$
- Return best of 3 solutions.

$\delta = \min(12, 21)$
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.
  - Only check distances of those within 11 positions in sorted list!

Def. Let $s_i$ be the point in the 2δ-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 δ.

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 \( \delta \). }

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)**

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.

\[
\begin{array}{cccccccc}
  1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
  \times & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
  \hline \\
  1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
  0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
  1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
  \end{array}
\]

Add: 11111101 01111111 11010101 00000000 11010101 11010101 11010100 00000000 01101000 00000000 00010000
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.

\[
x = 2^{n/2}x_1 + x_0 \\
y = 2^{n/2}y_1 + y_0 \\
x'y = (2^{n/2}x_1 + x_0)(2^{n/2}y_1 + y_0) = 2^n(x_1y_1 + x_0y_0) + 2^{n/2}(x_1y_0 + x_0y_1) + x_0y_0
\]

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

\[
T(n) = 4T(n/2) + \Theta(n) \Rightarrow T(n) = O(n^{1.585})
\]

Assumes \( n \) is a power of 2.
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?

## Key Idea

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

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

**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.

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

**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.

$$
T(n) = 7T(n/2) + \Theta(n^3) \quad \Rightarrow \quad T(n) = \Theta(n^{\log_27}) = O(n^{\log_27})
$$
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 64 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

Q. Multiply two 2-by-2 matrices with only 7 scalar multiplications?
A. Yes! [Strassen, 1969] \( \Theta(n^{3/2}) = O(n^{2.5}) \)

Q. Multiply two 2-by-2 matrices with only 6 scalar multiplications?
A. Impossible. [Hopcroft and Kerr, 1971] \( \Theta(n^{3/2}) = O(n^{2.75}) \)

Q. Two 3-by-3 matrices with only 21 scalar multiplications?
A. Also impossible. \( \Theta(n^{3/2}) = O(n^{2.75}) \)

Q. Two 70-by-70 matrices with only 143,640 scalar multiplications?
A. Yes! [Pan, 1980] \( \Theta(n^{10/9 \cdot 143640}) = O(n^{2.80}) \)

Decimal wars.
- December, 1979: \( O(n^{2.521813}) \).
- January, 1980: \( O(n^{2.521801}) \).

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

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

Caveat. Theoretical improvements to Strassen are progressively less practical.