Chapter 5
Divide and Conquer

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

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

Sort:

- 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.

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.

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.

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

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

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

Solution. \( T(n) = O(n \log_2 n) \).

A Useful Recurrence Relation

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

Mergesort recurrence.

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.

Assorted proofs. We describe several ways to prove this recurrence. Initially we assume \( n \) is a power of 2 and replace < with =.

Jon von Neumann (1945)
Proof by Telescoping

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

**Pf.** (by telescoping)

\[
T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
\frac{2T(n/2)}{n} + \frac{n}{\log_2 n} & \text{otherwise}
\end{cases}
\]

For \( n > 1 \):

\[
\begin{align*}
T(n) &= \frac{2T(n/2)}{n} + 1 \\
&= \frac{T(n/2)}{n/2} + 1 \\
&= \frac{T(n/4)}{n/4} + 1 + 1 \\
&= \cdots \\
&= \frac{T(n/n)}{n/n} + 1 + \cdots + 1 \\
&= \log_2 n
\end{align*}
\]

Proof by Induction

**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 \( T(2n) = 2n \log_2 (2n) \).

\[
\begin{align*}
T(2n) &= 2T(n) + 2n \\
&= 2n \log_2 n + 2n \\
&= 2n(\log_2 (2n) - 1) + 2n \\
&= 2n \log_2 (2n)
\end{align*}
\]

Analysis of Mergesort Recurrence

**Claim.** If \( T(n) \) satisfies the following recurrence, then \( T(n) = n \lceil \log_2 n \rceil \).

**Pf.** (by induction on \( n \))

- **Base case:** \( n = 1 \).
- **Define** \( n_1 = \lfloor n / 2 \rfloor \), \( n_2 = \lceil n / 2 \rceil \).
- **Induction step:** assume true for \( 1, 2, \ldots, n-1 \).

\[
\begin{align*}
T(n) &\leq \lfloor \lg n_1 \rfloor + \lceil \lg n_2 \rceil + n \\
&\leq \lfloor \lg n_2 \rfloor + \lfloor \lg n_2 \rfloor + n \\
&= \lceil \lg n \rceil + n \\
&\leq n \lceil \lg n \rceil - 1 + n \\
&= n \lceil \lg n \rceil - 1
\end{align*}
\]

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: $1, 2, \ldots, n$.
- 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 $j$.

### Counting Inversions

**Songs**

<table>
<thead>
<tr>
<th></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>

**Inversions**

3-2, 4-2

### 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.**

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

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

**Divide:** $O(1)$.
**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:** \( 2T(n/2) \).

\[
\begin{align*}
5-4, & \quad 5-2, \quad 4-2, \quad 8-2, \quad 10-2 \\
5-3, & \quad 4-3, \quad 8-6, \quad 8-3, \quad 8-7, \quad 10-6, \quad 10-9, \quad 10-3, \quad 10-7
\end{align*}
\]

**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.

\[
\begin{align*}
3 & \quad 7 & \quad 10 & \quad 14 & \quad 18 & \quad 19 \\
2 & \quad 11 & \quad 16 & \quad 17 & \quad 20 & \quad 25
\end{align*}
\]

13 blue-green inversions: \( 6 + 3 + 2 + 0 + 0 \)  
**Count:** \( O(n) \)

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

**Merge:** \( O(n) \)

---

**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
  \( r_A, A \) ← Sort-and-Count(A)
  \( r_B, B \) ← Sort-and-Count(B)
  \( r, L \) ← Merge-and-Count(A, B)

return \( r = r_A + r_B + r \) and the sorted list L
```

---

**T(n) \leq T\left(\left\lfloor n/2 \right\rfloor\right) + T\left(\left\lceil n/2 \right\rceil\right) + O(n) \Rightarrow 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.
  fast closest pair inspired 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.

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 $\frac{1}{3}n$ points on each side.
- **Conquer**: find closest pair in each side recursively.
- **Combine**: find closest pair with one point in each side, assuming that distance $< \delta$.
- 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.

Next slide

Closest Pair of Points

Sort points in 2δ-strip by their y coordinate.

- Only check distances of those within 11 positions in sorted list!

Next slide

Closest Pair of Points

Def. Let si be the point in the 2δ-strip, with the ith smallest y-coordinate.

Claim. If |i - j| ≥ 12, then the distance between si and sj is at least δ.

Pf. 
- No two points lie in same ½δ-by-½δ box.
- Two points at least 2 rows apart have distance ≥ 2(½δ).

Fact. Still true if we replace 12 with 7.
5.5 Integer Multiplication

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.

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

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: \( O(n^2) \) bit operations.
Divide-and-Conquer Multiplication: Warmup

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.

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

**Theorem.** [Karatsuba-Ofman, 1962] Can multiply two n-digit integers in \(O(n^{\log_23}) = O(n^{1.585})\) bit operations.

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.

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

**Theorem.** [Karatsuba-Ofman, 1962] Can multiply two n-digit integers in \(O(n^{\log_23}) = O(n^{1.585})\) bit operations.

Matrix Multiplication
Matrix Multiplication

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

![Matrix multiplication formula]

Brute force. \( \Theta(n^3) \) arithmetic operations.

Fundamental question. Can we improve upon brute force?

Matrix Multiplication: Key Idea

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

![Key idea matrix multiplication]

- 7 multiplications.
- 18 = 10 + 8 additions (or subtractions).

Fast Matrix Multiplication

Fast matrix multiplication. (Strassen, 1969)

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

Analysis.

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

![Fast matrix multiplication]

Matrix Multiplication: Warmup

Divide-and-conquer.

- Divide: partition \( A \) and \( B \) into \( \frac{1}{3}n \times \frac{1}{3}n \) blocks.
- Conquer: multiply 8 \( \frac{1}{3}n \times \frac{1}{3}n \) recursively.
- Combine: add appropriate products using 4 matrix additions.

![Warmup matrix multiplication]

\( T(n) = 8T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^3) \)
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 a 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, and other matrix ops.

Fast Matrix Multiplication in Theory

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

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

Caveat. Theoretical improvements to Strassen are progressively less practical.

Q. Multiply two 2-by-2 matrices with only 7 scalar multiplications?
A. Yes! [Strassen, 1969] \( \Theta(n^{2+}) = O(n^{2.41}) \)

Q. Multiply two 2-by-2 matrices with only 6 scalar multiplications?
A. Impossible. [Hopcroft and Kerr, 1971] \( \Theta(n^{4+}) = O(n^{2.90}) \)

Q. Two 3-by-3 matrices with only 21 scalar multiplications?
A. Also impossible. \( \Theta(n^{6+}) = O(n^{1.77}) \)

Q. Two 70-by-70 matrices with only 143,640 scalar multiplications?
A. Yes! [Pan, 1980] \( \Theta(n^{log_{2}(143640)}) = O(n^{2.81}) \)

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