Chapter 5

Divide and Conquer

CLRS 4.3
Divide-and-Conquer

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: $n \log n$.

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

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.

**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.

...
Mergesort.

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

Jon von Neumann (1945)
Merging

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. [Kronrod, 1969]

using only a constant amount of extra storage
Merging

Merge.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.
Merging

Merge.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

```
smallest
A G L O R
smallest
H I M S T
```

```
A G
auxiliary array
```
Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

```
A   G   L   O   R
H   I   M   S   T
```

Auxiliary array

```
A   G   H
```
Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

A   G   L   O   R

H   I   M   S   T

auxiliary array
Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

Auxiliary array: 

```
A G L O R
H I M S T
```

```
A G H I L
```

smallest

- blue arrow

- green arrow

Merging
Merging

Merge.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.
Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

![Diagram showing merging process with auxiliary array]
Merging

Merge.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

Merge

- A G L O R
- H I M S T

Auxiliary array

- A G H I L M O R
Merging

Merge.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.
Merging

Merge.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

```
A G L O R
H I M S T
```

```
A G H I L M O R S T
```
Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

A G L O R | H I M S T
---|---
first half exhausted | second half exhausted

AGHILMORSST

auxiliary array
Def. $T(n) = \text{number of comparisons to mergesort an input of size } n$. 

Mergesort recurrence.

Solution. $T(n) \in O(n \log_2 n)$.

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

\[
T(n) \le \begin{cases} 
0 & \text{if } n = 1 \\
T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + T\left(\left\lceil \frac{n}{2} \right\rceil \right) + n & \text{otherwise}
\end{cases}
\]
Proof by Recursion Tree

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

\[ T(n) \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]

\[ \text{T(n/2)} \]

\[ \text{T(n/4)} \]

\[ \text{T(2)} \]
Claim. If \( T(n) \) satisfies this recurrence, then \( T(n) = n \log_2 n \).

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

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

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

\[
T(2n) = 2T(n) + 2n \\
= 2n \log_2 n + 2n \\
= 2n(\log_2 (2n) - 1) + 2n \\
= 2n \log_2 (2n)
\]
Claim. If \( T(n) \) satisfies the following recurrence, then \( T(n) \leq n \lfloor \lg n \rfloor \).

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

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

- Base case: \( n = 1 \). \( T(1) = 0 = 1 \lfloor \lg 1 \rfloor \).
- Define \( n_1 = \lceil n/2 \rceil \), \( n_2 = \lfloor n/2 \rfloor \).
- Induction step: Let \( n \geq 2 \), assume true for 1, 2, ..., \( n-1 \).

\[
\begin{align*}
T(n) & \leq T(n_1) + T(n_2) + n \\
& \leq n_1 \lfloor \lg n_1 \rfloor + n_2 \lfloor \lg n_2 \rfloor + n \\
& \leq n_1 \lfloor \lg n_2 \rfloor + n_2 \lfloor \lg n_2 \rfloor + n \\
& = n \lfloor \lg n_2 \rfloor + n \\
& \leq n(\lfloor \lg n \rfloor - 1) + n \\
& = n \lfloor \lg n \rfloor
\end{align*}
\]
5.4 Closest Pair of Points
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.

\[\text{to make presentation cleaner}\]
Closest Pair of Points

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.
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**
Closest Pair of Points

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

\[ \delta = \min(12, 21) \]
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 $L$. 

$\delta = \min(12, 21)$
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

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!

\[ \delta = \min(12, 21) \]
**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 $\geq 2(\frac{1}{2}\delta) = \delta$.

**Fact.** Still true if we replace 12 with 7.

Scan points in $y$-order and compare distance between each point and next 11 neighbours. If any of these distances is less than $\delta$, update $\delta$. 
Smallest-Dist(p₁, ..., pₙ) {

    if n=2 then return dist(p₁,p₂)

    Compute separation line L such that half the points are on one side and half on the other side.

    δ' = Smallest-Dist(left half)
    δ'' = Smallest-Dist(right half)
    δ = min(δ',δ'')

    Delete all points further than δ from separation line L

    Sort remaining points by y-coordinate.

    Scan points in y-order and compare distance between each point and next 11 neighbours. If any of these distances is less than δ, update δ.

    return δ.
}
Closest Pair of Points

Closest-Pair\( (p_1, \ldots, p_n) \) { 

if \( n=2 \) then return dist\( (p_1, p_2) \), \( p_1, p_2 \)

Compute separation line \( L \) such that half the points are on one side and half on the other side.

\( \delta', p', q' = \text{Closest-Pair(left half)} \)
\( \delta'', p'', q'' = \text{Closest-Pair(right half)} \)
\( \delta, p, q = \min(\delta', \delta'') (p', q', p'', q'') \)

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 neighbours. If any of these distances is less than \( \delta \), update \( \delta, p, q \).

return \( \delta, p, q \).
}
Closest Pair of Points: Analysis

Running time.

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

**Q.** Can we achieve \( O(n \log n) \)?

**A.** Yes. First sort all points according to \( x \) coordinate before algo. Don’t sort points in strip from scratch each time.
- Each recursion returns a list: all points sorted by \( y \) coordinate.
- Sort by **merging** two pre-sorted lists.

\[ T(n) \leq 2T(n/2) + O(n) \Rightarrow T(n) \in O(n \log n) \]
Matrix Multiplication
Matrix Multiplication

Matrix multiplication. Given two n-by-n matrices $A$ and $B$, compute $C = AB$.

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

$$
\begin{bmatrix}
    c_{11} & c_{12} & \cdots & c_{1n} \\
    c_{21} & c_{22} & \cdots & c_{2n} \\
    \vdots & \vdots & \ddots & \vdots \\
    c_{n1} & c_{n2} & \cdots & c_{nn}
\end{bmatrix} = 
\begin{bmatrix}
    a_{11} & a_{12} & \cdots & a_{1n} \\
    a_{21} & a_{22} & \cdots & a_{2n} \\
    \vdots & \vdots & \ddots & \vdots \\
    a_{n1} & a_{n2} & \cdots & a_{nn}
\end{bmatrix} \times 
\begin{bmatrix}
    b_{11} & b_{12} & \cdots & b_{1n} \\
    b_{21} & b_{22} & \cdots & b_{2n} \\
    \vdots & \vdots & \ddots & \vdots \\
    b_{n1} & b_{n2} & \cdots & b_{nn}
\end{bmatrix}
$$

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 \( \frac{1}{2}n \times \frac{1}{2}n \) blocks.
- **Conquer:** multiply \( 8 \frac{1}{2}n \times \frac{1}{2}n \) recursively.
- **Combine:** add appropriate products using 4 matrix additions.

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

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

\[
T(n) = 8T(n/2) + \Theta(n^2) \quad \Rightarrow \quad T(n) \in \Theta(n^3)
\]
Matrix Multiplication: Key Idea

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

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

\[
C_{11} = P_5 + P_4 - P_2 + P_6 \\
C_{12} = P_1 + P_2 \\
C_{21} = P_3 + P_4 \\
C_{22} = P_5 + P_1 - P_3 - P_7
\]

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

- 7 multiplications.
- 18 = 10 + 8 additions (or subtractions).
Strassen: Recursion Tree

\[
T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
7T(n/2) + n^2 & \text{otherwise}
\end{cases}
\]

\[
T(n) = \sum_{k=0}^{\log_2 n} n^2 \left(\frac{7}{4}\right)^k = n^2 \frac{\left(\frac{7}{4}\right)^{1+\log_2 n} - 1}{\frac{7}{4} - 1} \approx \frac{7}{3} n^{\log_2 7}.
\]
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 18 matrix additions.

Analysis.

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

\[
T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2) \quad \Rightarrow \quad T(n) \in \Theta(n^{\log_2 7}) \in O(n^{2.81})
\]
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

Q. Multiply two 2-by-2 matrices with only 7 scalar multiplications?
A. Yes! [Strassen, 1969] \( \Theta(n^{\log_2 7}) \in O(n^{2.81}) \)

Q. Multiply two 2-by-2 matrices with only 6 scalar multiplications?
A. Impossible. [Hopcroft and Kerr, 1971] \( \Theta(n^{\log_2 6}) \in O(n^{2.59}) \)

Q. Two 3-by-3 matrices with only 21 scalar multiplications?
A. Also impossible. \( \Theta(n^{\log_3 21}) \in O(n^{2.77}) \)

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

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

**Best known.** $O(n^{2.376})$ [Coppersmith-Winograd, 1987-2010.]

In 2010, Andrew Stothers gave an improvement to the algorithm $O(n^{2.374})$. In 2011, Virginia Williams combined a mathematical short-cut from Stothers' paper with her own insights and automated optimization on computers, improving the bound $O(n^{2.3728642})$. In 2014, François Le Gall simplified the methods of Williams and obtained an improved bound of $O(n^{2.3728639})$.

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

**Caveat.** Theoretical improvements to Strassen are progressively less practical (hidden constant gets worse).
CLRS 4.3 Master Theorem
Master Theorem from CLRS 4.3

Used for many divide-and-conquer recurrences

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$, $b > 1$, and $f(n) > 0$.

$a = \text{(constant) number of sub-instances},$
$b = \text{(constant) size ratio of sub-instances},$
$f(n) = \text{time used for dividing and recombining}.$

Based on the master theorem (Theorem 4.1).
Compare $n^{\log_b a}$ vs. $f(n)$:
Proof by Recursion Tree

Used for many divide-and-conquer recurrences

\[ T(n) = aT(n/b) + f(n) \]

where \( a \geq 1, b > 1, \) and \( f(n) > 0. \)
\[ T(n) = aT(n/b) + f(n) \]

**Case 1:** \( f(n) \in O(n^L) \) for some constant \( L < \log_b a \).

**Solution:** \( T(n) \in \Theta(n^{\log_b a}) \)

**Case 2:** \( f(n) \in \Theta(n^{\log_b a \log^k n}) \), for some \( k \geq 0 \).

**Solution:** \( T(n) \in \Theta(n^{\log_b a \log^{k+1} n}) \)

**Case 3:** \( f(n) \in \Omega(n^L) \) for some constant \( L > \log_b a \) and \( f(n) \) satisfies the regularity condition \( af(n/b) \leq cf(n) \) for some \( c < 1 \) and all large \( n \).

**Solution:** \( T(n) \in \Theta(f(n)) \)
Master Theorem

\[ T(n) = aT(n/b) + f(n) \]

where \( a \geq 1, b > 1, \) and \( f(n) > 0. \)

**Case 1:** \( f(n) \in O(n^L) \) for some constant \( L < \log_b a. \) (\( f(n) \) is polynomially smaller than \( n^{\log_b a} \).)

**Solution:** \( T(n) \in \Theta(n^{\log_b a}) \)

(Intuitively: cost is dominated by leaves.)
Case 1: \( f(n) \in O(n^L) \) for some constant \( L < \log_b a \).

Solution: \( T(n) \in \Theta(n^{\log_b a}) \)

\[
T(n) = 5T(n/2) + \Theta(n^2)
\]

Compare \( n^{\log_2 5} \) vs. \( n^2 \).

Since \( 2 < \log_2 5 \) use Case 1

Solution: \( T(n) \in \Theta(n^{\log_2 5}) \)
Master Theorem

\[ T(n) = aT(n/b) + f(n) \]

where \( a \geq 1, b > 1, \) and \( f(n) > 0. \)

**Simple Case 2:** \( f(n) \in \Theta(n^{\log_b a}). \)

**Solution:** \( T(n) \in \Theta(n^{\log_b a \log n}) \)
Master Theorem

\[ T(n) = aT(n/b) + f(n) \]

where \( a \geq 1, \ b > 1, \) and \( f(n) > 0. \)

Case 2: \( f(n) \in \Theta(n^\log_b a \log^k n), \) for some \( k \geq 0. \)

Solution: \( T(n) \in \Theta(n^\log_b a \log^{k+1} n) \)

(Intuitively: cost is \( n^\log_b a \lg^k n \) at each level, and there are \( \Theta(\lg n) \) levels.)
Case 2: \( f(n) \in \Theta(n^{\log_b a} \log^k n) \), for some \( k \geq 0 \).

Solution: \( T(n) \in \Theta(n^{\log_b a} \log^{k+1} n) \)

\[
T(n) = 27T(n/3) + \Theta(n^3 \log n)
\]

Compare \( n^{\log_3 27} \) vs. \( n^3 \).

Since \( 3 = \log_3 27 \) use Case 2

Solution: \( T(n) \in \Theta(n^3 \log^2 n) \)
Master Theorem

\[ T(n) = aT\left(\frac{n}{b}\right) + f(n) \]
where \( a \geq 1, b > 1, \) and \( f(n) > 0. \)

Case 3: \( f(n) \in \Omega(n^L) \) for some constant \( L > \log_b a \)
and \( f(n) \) satisfies the regularity condition \( af(n/b) \leq cf(n) \) for some \( c < 1 \) and all large \( n. \)
\( (f(n) \) is polynomially greater than \( n^{\log_b a}. \)\)

Solution: \( T(n) \in \Theta(f(n)) \)
(Intuitively: cost is dominated by root.)
Master Theorem

**Case 3:** \( f(n) \in \Omega(n^L) \) for some constant \( L > \log_b a \)
and \( f(n) \) satisfies the regularity condition \( af(n/b) \leq cf(n) \) for some \( c<1 \) and all large \( n \).

**Solution:** \( T(n) \in \Theta(f(n)) \)

---

**What’s with the Case 3 regularity condition?**

- Generally not a problem.
- It always holds whenever \( f(n) = n^k \) and \( f(n) \in \Omega(n^{\log_b a + \epsilon}) \) for constant \( \epsilon > 0 \).
**Master Theorem**

**Case 3:** $f(n) \in \Omega(n^L)$ for some constant $L > \log_b a$
and $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some $c<1$ and all large $n$.

**Solution:** $T(n) \in \Theta(f(n))$

\[
T(n) = 5T(n/2) + \Theta(n^3)
\]

Compare $n^{\log_2 5}$ vs. $n^3$.

Since $3 > \log_2 5$ use **Case 3**

$af(n/b) = 5(n/2)^3 = 5/8 \ n^3 \leq cn^3$, for $c = 5/8$

**Solution:** $T(n) \in \Theta(n^3)$
Master Theorem

\[ T(n) = 27T(n/3) + \Theta(n^3/\log n) \]

Compare \( n^{\log_3 27} \) vs. \( n^3 \).

Since \( 3 = \log_3 27 \) use Case 2

but \( n^3/\log n \in \text{not } \Theta(n^3 \log^k n) \) for \( k \geq 0 \)

Cannot use Master Method.
Beyond the Master Theorem

Fundamentals of Algorithmics

Gilles Brassard
Paul Bratley
Median Finding
Median Finding. Given \( n \) distinct numbers \( a_1, \ldots, a_n \), find \( i \) such that

\[
|\{ j : a_j < a_i \}| = \lfloor n-1 / 2 \rfloor \quad \text{and} \quad |\{ j : a_j > a_i \}| = \lceil n-1 / 2 \rceil .
\]

\[
\begin{array}{cccccccccc}
22 & 31 & 44 & 7 & 12 & 19 & 20 & 35 & 3 & 40 & 27 \\
\end{array}
\]

\[
\begin{array}{cccccccccc}
3 & 7 & 12 & 19 & 20 & 22 & 27 & 31 & 35 & 40 & 44 \\
9 & 4 & 5 & 6 & 7 & 1 & 11 & 2 & 8 & 10 & 3 \\
\end{array}
\]
Selection

Selection. Given \( n \) distinct numbers \( a_1, \ldots, a_n \), and index \( k \), find \( i \) such that

\[
|\{ j : a_j < a_i \}| = k-1 \text{ and } |\{ j : a_j > a_i \}| = n-k.
\]

\( k=4 \)

\[
\begin{array}{cccccccccc}
22 & 31 & 44 & 7 & 12 & 19 & 20 & 35 & 3 & 40 & 27 \\
\end{array}
\]

\[
\begin{array}{cccccccccc}
3 & 7 & 12 & 19 & 20 & 22 & 27 & 31 & 35 & 40 & 44 \\
9 & 4 & 5 & 6 & 7 & 1 & 11 & 2 & 8 & 10 & 3 \\
\end{array}
\]

\[
\text{Median}( a_1, \ldots, a_n ) = \text{Selection}( a_1, \ldots, a_n, \lfloor n+1 / 2 \rfloor )
\]
Algorithm partition(A, start, stop)
Input: An array A, indices start and stop.
pivot ← A[stop]
left ← start
right ← stop - 1
while left ≤ right do
  while left ≤ right and A[left] ≤ pivot do left ← left + 1
  while (left ≤ right and A[right] ≥ pivot) do right ← right -1
  if (left < right ) then exchange A[left] ↔ A[right]
return left
Example of execution of partition

\[ A = \begin{bmatrix} 6 & 3 & 7 & 3 & 2 & 5 & 7 & 5 \end{bmatrix} \quad \text{pivot} = 5 \]

\[ A = \begin{bmatrix} 6 & 3 & 7 & 3 & 2 & 5 & 7 & 5 \end{bmatrix} \quad \text{swap 6, 2} \]

\[ A = \begin{bmatrix} 2 & 3 & 7 & 3 & 6 & 5 & 7 & 5 \end{bmatrix} \]

\[ A = \begin{bmatrix} 2 & 3 & 7 & 3 & 6 & 5 & 7 & 5 \end{bmatrix} \quad \text{swap 7,3} \]

\[ A = \begin{bmatrix} 2 & 3 & 3 & 7 & 6 & 5 & 7 & 5 \end{bmatrix} \]

\[ A = \begin{bmatrix} 2 & 3 & 3 & 7 & 6 & 5 & 7 & 5 \end{bmatrix} \quad \text{swap 7,pivot} \]

\[ A = \begin{bmatrix} 2 & 3 & 3 & 5 & 6 & 5 & 7 & 7 \end{bmatrix} \]
Selection from Median

**Selection.** Given \( n \) distinct numbers \( a_1, \ldots, a_n \), and index \( k \), find \( i \) such that

\[
|\{ j : a_j < a_i \}| = k-1 \quad \text{and} \quad |\{ j : a_j > a_i \}| = n-k.
\]

\[ k = \lfloor \frac{n+1}{2} \rfloor \quad \text{return} \quad k \]
**Selection**

**Selection.** Given \( n \) distinct numbers \( a_1, \ldots, a_n \), and index \( k \), find \( i \) such that

\[
|\{ j : a_j < a_i \}| = k - 1 \quad \text{and} \quad |\{ j : a_j > a_i \}| = n - k.
\]

**Pseudo-Median**

\[
\text{Selection}(a_1, \ldots, a_m, k) \quad \text{Selection}(a_{m+1}, \ldots, a_n, k-m)
\]

**Partition**

\( k = m \) return \( k \)
Selection. Given \( n \) distinct numbers \( a_1, \ldots, a_n \), and index \( k \), find \( i \) such that

\[
|\{ j : a_j < a_i \}| = k-1 \quad \text{and} \quad |\{ j : a_j > a_i \}| = n-k.
\]
Selection

Selection. Given n distinct numbers $a_1, \ldots, a_n$, and index $k$, find $i$ such that

$$|\{ j : a_j < a_i \}| = k - 1 \text{ and } |\{ j : a_j > a_i \}| = n - k.$$  

$$T(n) \leq T\left(\lceil n/5 \rceil \right) + \max \{ T\left(\lceil 3n/10 \rceil \right), \ldots, T\left(\lceil 7n/10 \rceil \right) \} + \Theta(n)$$

Solution: $T(n) \in \Theta(n)$
Selection

Selection. Given $n$ distinct numbers $a_1, \ldots, a_n$, and index $k$, find $i$ such that

$$|\{ j : a_j < a_i \}| = k-1 \text{ and } |\{ j : a_j > a_i \}| = n-k.$$

$$T(n) \leq T(\ n/5\ ) + T(\ 7n/10\ ) + \Theta(n)$$

Solution: $T(n) \in \Theta(n)$

Assuming $T(i) \leq d \cdot i$ for $1 \leq i \leq n$, $\Theta(n) \leq cn$

$T(n+1) \leq T(n+1/5) + T(7(n+1)/10) + c(n+1)$

$\leq d(n+1)/5 + 7d(n+1)/10 + c(n+1)$

$= (2d+7d+10c)/10\ (n+1)$

$= (9d+10c)/10\ (n+1)$

$\leq d\ (n+1)$ \quad \text{as long as } (9d+10c)/10 \leq d, \text{ or equivalently } 10c \leq d.$
Selection

Selection. Given n distinct numbers $a_1, \ldots, a_n$, and index $k$, find $i$ such that

$$\{| j : a_j < a_i \} | = k-1 \text{ and } | \{ j : a_j > a_i \} | = n-k.$$ 

$$T(n) \leq T\left( \frac{n}{5} \right) + T\left( \frac{7n}{10} \right) + \Theta(n)$$

Solution: $T(n) \in \Theta(n)$

**Example:** $d=10c$,

Assuming $T(i) \leq 10c$ i for $1 \leq i \leq n$, $\Theta(n) \leq cn$

$$T(n+1) \leq T\left( \frac{n+1}{5} \right) + T\left( \frac{7}{10} (n+1) \right) + c(n+1)$$

$$\leq 10c/5 \ (n+1) + 7\cdot 10c/10 \ (n+1) + c(n+1)$$

$$= (2c+7c+c)(n+1)$$

$$= 10c \ (n+1)$$