5. **DIVIDE AND CONQUER I**

- mergesort
- counting inversions
- closest pair of points
- randomized quicksort
- median and selection
Divide-and-conquer paradigm

Divide-and-conquer.

- Divide up problem into several subproblems.
- Solve each subproblem recursively.
- Combine solutions to subproblems into overall solution.

Most common usage.

- Divide problem of size $n$ into two subproblems of size $n/2$ in linear time.
- Solve two subproblems recursively.
- Combine two solutions into overall solution in linear time.

Consequence.

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

attributed to Julius Caesar
5. Divide and Conquer

- mergesort
- counting inversions
- closest pair of points
- randomized quicksort
- median and selection
Sorting problem

Problem. Given a list of $n$ elements from a totally-ordered universe, rearrange them in ascending order.
Sorting applications

Obvious applications.
  • Organize an MP3 library.
  • Display Google PageRank results.
  • List RSS news items in reverse chronological order.

Some problems become easier once elements are sorted.
  • Identify statistical outliers.
  • Binary search in a database.
  • Remove duplicates in a mailing list.

Non-obvious applications.
  • Convex hull.
  • Closest pair of points.
  • Interval scheduling / interval partitioning.
  • Minimum spanning trees (Kruskal's algorithm).
  • Scheduling to minimize maximum lateness or average completion time.
  • ...
Mergesort

- Recursively sort left half.
- Recursively sort right half.
- Merge two halves to make sorted whole.

<table>
<thead>
<tr>
<th>Input</th>
<th>ALGORITYHSMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort left half</td>
<td>AGLORITHTHMS</td>
</tr>
<tr>
<td>Sort right half</td>
<td>AGLORHMIST</td>
</tr>
<tr>
<td>Merge results</td>
<td>AGHILMORSST</td>
</tr>
</tbody>
</table>
Merging

**Goal.** Combine two sorted lists $A$ and $B$ into a sorted whole $C$.

- Scan $A$ and $B$ from left to right.
- Compare $a_i$ and $b_j$.
- If $a_i \leq b_j$, append $a_i$ to $C$ (no larger than any remaining element in $B$).
- If $a_i > b_j$, append $b_j$ to $C$ (smaller than every remaining element in $A$).

- sorted list $A$
  
  | 3 | 7 | 10 | $a_i$ | 18 |
  |

- sorted list $B$
  
  | 2 | 11 | $b_j$ | 17 | 23 |
  |

merge to form sorted list $C$

| 2 | 3 | 7 | 10 | 11 |
A useful recurrence relation

**Def.** \( T(n) = \) max number of compares to mergesort a list of size \( \leq n \).

**Note.** \( T(n) \) is monotone nondecreasing.

**Mergesort recurrence.**

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

**Solution.** \( T(n) \) is \( O(n \log_2 n) \).

**Assorted proofs.** We describe several ways to prove this recurrence. Initially we assume \( n \) is a power of 2 and replace \( \leq \) with \( = \).
Divide-and-conquer recurrence: proof by recursion tree

**Proposition.** If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

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

**Pf 1.**

\[ T(n) = \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots + n \log_2 n \]

assuming $n$ is a power of 2
Proof by induction

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

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

Pf 2. [by induction on $n$]

- Base case: when $n = 1$, $T(1) = 0$.
- Inductive hypothesis: assume $T(n) = n \log_2 n$.
- Goal: show that $T(2n) = 2n \log_2 (2n)$.

$$
T(2n) = 2 \, T(n) + 2n
= 2 \, n \log_2 n + 2n
= 2 \, n \, (\log_2 (2n) - 1) + 2n
= 2 \, n \log_2 (2n).
$$

assuming $n$ is a power of 2
Analysis of mergesort recurrence

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

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

Pf. [by strong induction on \( n \)]
- Base case: \( n = 1 \).
- Define \( n_1 = \lfloor n/2 \rfloor \) and \( n_2 = \lceil n/2 \rceil \).
- Induction step: assume true for \( 1, 2, \ldots, n - 1 \).

\[
T(n) \leq T(n_1) + T(n_2) + n \\
\leq n_1 \lceil \log_2 n_1 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\
\leq n_1 \lceil \log_2 n_2 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\
= n \lceil \log_2 n_2 \rceil + n \\
\leq n (\lceil \log_2 n \rceil - 1) + n \\
= n \lceil \log_2 n \rceil.
\]
Section 5.3

5. Divide and Conquer

- mergesort
- counting inversions
- closest pair of points
- randomized quicksort
- median and selection
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 $j$ are inverted if $i < j$, but $a_i > a_j$.

<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>

2 inversions: 3–2, 4–2

Brute force: check all $\Theta(n^2)$ pairs.
Counting inversions: 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).

---

**Rank Aggregation Methods for the Web**

Cynthia Dwork† Ravi Kumar† Moni Naor† D. Sivakumar†

**ABSTRACT**

We consider the problem of combining ranking results from various sources. In the context of the Web, the main applications include building meta-search engines, combining ranking functions, selecting documents based on multiple criteria, and improving search precision through word associations. We develop a set of techniques for the rank aggregation problem and compare their performance to that of well-known methods. A primary goal of our work is to design rank aggregation techniques that can effectively combat "spam," a serious problem in Web searches. Experiments show that our methods are simple, efficient, and effective.

**Keywords:** rank aggregation, ranking functions, meta-search, multi-word queries, spam
Counting inversions: divide-and-conquer

- Divide: separate list into two halves $A$ and $B$.
- Conquer: recursively count inversions in each list.
- Combine: count inversions $(a, b)$ with $a \in A$ and $b \in B$.
- Return sum of three counts.

**input**

<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>3</th>
<th>7</th>
</tr>
</thead>
</table>

**count inversions in left half A**

<table>
<thead>
<tr>
<th>1</th>
<th>5</th>
<th>4</th>
<th>8</th>
<th>10</th>
</tr>
</thead>
</table>

5-4

**count inversions in right half B**

<table>
<thead>
<tr>
<th>2</th>
<th>6</th>
<th>9</th>
<th>3</th>
<th>7</th>
</tr>
</thead>
</table>

6-3 9-3 9-7

**count inversions $(a, b)$ with $a \in A$ and $b \in B$**

<table>
<thead>
<tr>
<th>1</th>
<th>5</th>
<th>4</th>
<th>8</th>
<th>10</th>
</tr>
</thead>
</table>

4-2 4-3 5-2 5-3 8-2 8-3 8-6 8-7 10-2 10-3 10-6 10-7 10-9

**output $1 + 3 + 13 = 17$**
Counting inversions: how to combine two subproblems?

Q. How to count inversions \((a, b)\) with \(a \in A\) and \(b \in B\)?
A. Easy if \(A\) and \(B\) are sorted!

Warmup algorithm.
- Sort \(A\) and \(B\).
- For each element \(b \in B\),
  - binary search in \(A\) to find how elements in \(A\) are greater than \(b\).

<table>
<thead>
<tr>
<th>list A</th>
<th>list B</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 10 18 3 14</td>
<td>17 23 2 11 16</td>
</tr>
</tbody>
</table>

sort A
| 3 7 10 14 18 |

sort B
| 2 11 16 17 23 |

binary search to count inversions \((a, b)\) with \(a \in A\) and \(b \in B\)

| 3 7 10 14 18 |
| 2 11 16 17 23 |
| 5 2 1 1 0 |
Counting inversions: how to combine two subproblems?

Count inversions \((a, b)\) with \(a \in A\) and \(b \in B\), assuming \(A\) and \(B\) are sorted.

- Scan \(A\) and \(B\) from left to right.
- Compare \(a_i\) and \(b_j\).
- If \(a_i < b_j\), then \(a_i\) is not inverted with any element left in \(B\).
- If \(a_i > b_j\), then \(b_j\) is inverted with every element left in \(A\).
- Append smaller element to sorted list \(C\).

\[
\begin{array}{cccc}
3 & 7 & 10 & a_i \\
2 & 11 & b_j & 17 & 23 \\
5 & 2 &
\end{array}
\]

merge to form sorted list \(C\)

\[
\begin{array}{cccc}
2 & 3 & 7 & 10 & 11 \\
\end{array}
\]
Counting inversions: divide-and-conquer algorithm implementation

**Input.** List $L$.

**Output.** Number of inversions in $L$ and sorted list of elements $L'$.

\[\text{SORT-AND-COUNT}(L)\]

**IF** list $L$ has one element

\[\text{RETURN } (0, L).\]

**DIVIDE** the list into two halves $A$ and $B$.

\[(r_A, A) \leftarrow \text{SORT-AND-COUNT}(A).\]
\[(r_B, B) \leftarrow \text{SORT-AND-COUNT}(B).\]
\[(r_{AB}, L') \leftarrow \text{MERGE-AND-COUNT}(A, B).\]

**RETURN** $(r_A + r_B + r_{AB}, L')$. 
Proposition. The sort-and-count algorithm counts the number of inversions in a permutation of size $n$ in $O(n \log n)$ time.

Pf. The worst-case running time $T(n)$ satisfies the recurrence:

$$T(n) = \begin{cases} 
\Theta(1) & \text{if } n = 1 \\
T(\lfloor n / 2 \rfloor) + T(\lceil n / 2 \rceil) + \Theta(n) & \text{otherwise}
\end{cases}$$
5. Divide and Conquer

- mergesort
- counting inversions
- closest pair of points
- randomized quicksort
- median and selection
Closest pair of points

Closest pair problem. Given $n$ points in the plane, find a pair of points with the 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
Closest pair of points

**Closest pair problem.** Given $n$ points in the plane, find a pair of points with the smallest Euclidean distance between them.

**Brute force.** Check all pairs with $\Theta(n^2)$ distance calculations.

**1d version.** Easy $O(n \log n)$ algorithm if points are on a line.

**Nondegeneracy assumption.** No two points have the same $x$-coordinate.
Closest pair of points: first attempt

**Sorting solution.**

- Sort by $x$-coordinate and consider nearby points.
- Sort by $y$-coordinate and consider nearby points.
Closest pair of points: first attempt

**Sorting solution.**
- Sort by $x$-coordinate and consider nearby points.
- Sort by $y$-coordinate and consider nearby points.
Closest pair of points: second attempt

**Divide.** Subdivide region into 4 quadrants.
Closest pair of points: second attempt

**Divide.** Subdivide region into 4 quadrants.
**Obstacle.** Impossible to ensure $n/4$ points in each piece.
Closest pair of points: divide-and-conquer algorithm

- **Divide:** draw vertical line \( L \) so that \( n/2 \) 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.

![Diagram showing divide-and-conquer algorithm](image)

- Seems like \( \Theta(N^2) \)
How to find closest pair with one point in each side?

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

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

\[ \delta = \min(12, 21) \]
How to find closest pair with one point in each side?

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!

\[ \delta = \min(12, 21) \]
How to find closest pair with one point in each side?

**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 $\geq 2\left(\frac{1}{2}\delta\right)$.

**Fact.** Claim remains true if we replace 12 with 7.
Closest pair of points: divide-and-conquer algorithm

**CLOSEST-PAIR** \((p_1, p_2, \ldots, p_n)\)

Compute separation line \(L\) such that half the points are on each side of the line.

\[\delta_1 \leftarrow \text{CLOSEST-PAIR} \text{ (points in left half).}\]

\[\delta_2 \leftarrow \text{CLOSEST-PAIR} \text{ (points in right half).}\]

\[\delta \leftarrow \min \{ \delta_1, \delta_2 \}.\]

Delete all points further than \(\delta\) from 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** \(\delta\).
Theorem. The divide-and-conquer algorithm for finding the closest pair of points in the plane can be implemented in $O(n \log^2 n)$ time.

$$T(n) = \begin{cases} 
\Theta(1) & \text{if } n = 1 \\
T(\lceil n / 2 \rceil) + T(\lfloor n / 2 \rfloor) + O(n \log n) & \text{otherwise}
\end{cases}$$

Lower bound. In quadratic decision tree model, any algorithm for closest pair (even in 1D) requires $\Omega(n \log n)$ quadratic tests.
Improved closest pair algorithm

Q. How to improve to $O(n \log n)$?
A. Yes. Don't sort points in strip from scratch each time.
   - Each recursive returns two lists: all points sorted by $x$-coordinate, and all points sorted by $y$-coordinate.
   - Sort by merging two pre-sorted lists.

Theorem. [Shamos 1975] The divide-and-conquer algorithm for finding the closest pair of points in the plane can be implemented in $O(n \log n)$ time.

Pf. $T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n) & \text{otherwise} \end{cases}$

Note. See Section 13.7 for a randomized $O(n)$ time algorithm.
5. Divide and Conquer

- mergesort
- counting inversions
- closest pair of points
- randomized quicksort
- median and selection
Randomized quicksort

3-way partition array so that:
- Pivot element \( p \) is in place.
- Smaller elements in left subarray \( L \).
- Equal elements in middle subarray \( M \).
- Larger elements in right subarray \( R \).

Recur in both left and right subarrays.

**RANDOMIZED-QUICKSORT** \((A)\)

**IF** list \( A \) has zero or one element

**RETURN.**

Pick pivot \( p \in A \) uniformly at random.

\((L, M, R) \leftarrow \text{PARTITION-3-WAY} \((A, a_i)\).\)

**RANDOMIZED-QUICKSORT**(\(L\)).

**RANDOMIZED-QUICKSORT**(\(R\)).
Analysis of randomized quicksort

**Proposition.** The expected number of compares to quicksort an array of $n$ distinct elements is $O(n \log n)$.

**Pf.** Consider BST representation of partitioning elements.

![BST representation of partitioning elements]

The original array of elements $A$:

| 7 | 6 | 12 | 3 | 11 | 8 | 9 | 1 | 4 | 10 | 2 | 13 | 5 |

First partitioning element (chosen uniformly at random): 9

First partitioning element in left subarray: 3

First partitioning element in right subarray: 11
An analysis of randomized quicksort

**Proposition.** The expected number of compares to quicksort an array of $n$ distinct elements is $O(n \log n)$.

**Pf.** Consider BST representation of partitioning elements.

- An element is compared with only its ancestors and descendants.
Analysis of randomized quicksort

**Proposition.** The expected number of compares to quicksort an array of \( n \) distinct elements is \( O(n \log n) \).

**Pf.** Consider BST representation of partitioning elements.
- An element is compared with only its ancestors and descendants.
Analysis of randomized quicksort

**Proposition.** The expected number of compares to quicksort an array of \( n \) distinct elements is \( O(n \log n) \).

**Pf.** Consider BST representation of partitioning elements.
- An element is compared with only its ancestors and descendants.
- \( \Pr[ a_i \text{ and } a_j \text{ are compared } ] = 2 / |j - i + 1| \).

Pr[2 and 8 compared] = 2/7
(compared if either 2 or 8 are chosen as partition before 3, 4, 5, 6 or 7)
Analysis of randomized quicksort

**Proposition.** The expected number of compares to quicksort an array of \( n \) distinct elements is \( O(n \log n) \).

**Pf.** Consider BST representation of partitioning elements.

- An element is compared with only its ancestors and descendants.
- \( \Pr[ a_i \text{ and } a_j \text{ are compared } ] = \frac{2}{|j - i + 1|} \).

- Expected number of compares

\[
\begin{align*}
\sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{2}{j - i + 1} &= 2 \sum_{i=1}^{N} \sum_{j=2}^{N-i+1} \frac{1}{j} \\
&\leq 2N \sum_{j=1}^{N} \frac{1}{j} \\
&\sim 2N \int_{x=1}^{N} \frac{1}{x} \, dx \\
&= 2N \ln N
\end{align*}
\]

**Remark.** Number of compares only decreases if equal elements.
5. **Divide and Conquer**

- mergesort
- counting inversions
- closest pair of points
- randomized quicksort
- median and selection
Median and selection problems

Selection. Given $n$ elements from a totally ordered universe, find $k^{th}$ smallest.

- Minimum: $k = 1$; maximum: $k = n$.
- Median: $k = \lfloor (n + 1) / 2 \rfloor$.
- $O(n)$ compares for min or max.
- $O(n \log n)$ compares by sorting.
- $O(n \log k)$ compares with a binary heap.

Applications. Order statistics; find the "top $k$"; bottleneck paths, ...

Q. Can we do it with $O(n)$ compares?
A. Yes! Selection is easier than sorting.
Quickselect

3-way partition array so that:
• Pivot element $p$ is in place.
• Smaller elements in left subarray $L$.
• Equal elements in middle subarray $M$.
• Larger elements in right subarray $R$.

Recur in one subarray—the one containing the $k^{th}$ smallest element.

**Quick-Select** ($A, k$)

Pick pivot $p \in A$ uniformly at random. 

$$(L, M, R) \leftarrow \text{Partition-3-Way} (A, p).$$

**IF** $k \leq |L|$ **RETURN** Quick-Select ($L, k$).

**ELSE IF** $k > |L| + |M|$ **RETURN** Quick-Select ($R, k - |L| - |M|$)

**ELSE** **RETURN** $p$.  

3-way partitioning can be done in-place (using n–1 compares)
Quickselect analysis

**Intuition.** Split candy bar uniformly \( \Rightarrow \) expected size of larger piece is \( \frac{3}{4} \).

\[
T(n) \leq T\left(\frac{3}{4}n\right) + n \Rightarrow T(n) \leq 4n
\]

**Def.** \( T(n, k) = \) expected \# compares to select \( k^{th} \) smallest in an array of size \( \leq n \).

**Def.** \( T(n) = \max_k T(n, k) \).

**Proposition.** \( T(n) \leq 4n \).

**Pf.** [by strong induction on \( n \)]

- Assume true for \( 1, 2, \ldots, n - 1 \).
- \( T(n) \) satisfies the following recurrence:

\[
T(n) \leq n + 2 / n \left[ T(n/2) + \ldots + T(n-3) + T(n-2) + T(n-1) \right]
\leq n + 2 / n \left[ 4n/2 + \ldots + 4(n-3) + 4(n-2) + 4(n-1) \right]
= n + 4 \left(3/4 \cdot n\right)
= 4n. \quad \blacksquare
\]

can assume we always recur on largest subarray since \( T(n) \) is monotonic and we are trying to get an upper bound
tiny cheat: sum should start at \( T(\lfloor n/2 \rfloor) \)
Selection in worst case linear time

**Goal.** Find pivot element $p$ that divides list of $n$ elements into two pieces so that each piece is **guaranteed** to have $\leq \frac{7}{10} n$ elements.

**Q.** How to find approximate median in linear time?
**A.** Recursively compute median of sample of $\leq \frac{2}{10} n$ elements.

$$T(n) = \begin{cases} 
\Theta(1) & \text{if } n = 1 \\
T(\frac{7}{10} n) + T(\frac{2}{10} n) + \Theta(n) & \text{otherwise}
\end{cases}$$

due to two subproblems of different sizes!
Choosing the pivot element

- Divide $n$ elements into $\lfloor n / 5 \rfloor$ groups of 5 elements each (plus extra).
Choosing the pivot element

- Divide $n$ elements into $\lfloor n / 5 \rfloor$ groups of 5 elements each (plus extra).
- Find median of each group (except extra).

\begin{itemize}
  \item 29 10 38 37 2 55 18 24 34 35 36
  \item 22 44 52 11 53 12 13 43 20 4 27
  \item 28 23 6 26 40 19 1 46 31 49 8
  \item 14 9 5 3 54 30 48 47 32 51 21
  \item 45 39 50 15 25 16 41 17 22 7
\end{itemize}

\[N = 54\]
Choosing the pivot element

- Divide \( n \) elements into \( \lceil n / 5 \rceil \) groups of 5 elements each (plus extra).
- Find median of each group (except extra).
- Find median of \( \lceil n / 5 \rceil \) medians recursively.
- Use median-of-medians as pivot element.

\( N = 54 \)
**Median-of-medians selection algorithm**

\[ \text{MOM-SELECT} (A, k) \]

\[ n \leftarrow |A|. \]

**If** \( n < 50 \) **RETURN** \( k^{th} \) smallest of element of \( A \) via mergesort.

Group \( A \) into \( \lfloor n / 5 \rfloor \) groups of 5 elements each (plus extra).

\( B \leftarrow \text{median of each group of 5}. \)

\( p \leftarrow \text{MOM-SELECT}(B, \lfloor n / 10 \rfloor) \) \( \text{median of medians} \)

\((L, M, R) \leftarrow \text{PARTITION-3-WAY} (A, p).\)

**If** \( k \leq |L| \) **RETURN** \( \text{MOM-SELECT} (L, k) \).

**Else If** \( k > |L| + |M| \) **RETURN** \( \text{MOM-SELECT} (R, k - |L| - |M|) \).

**Else** **RETURN** \( p \).
Analysis of median-of-medians selection algorithm

- At least half of 5-element medians $\leq p$. 

N = 54
Analysis of median-of-medians selection algorithm

- At least half of 5-element medians \( \leq p \).
- At least \( \lfloor n/5 \rfloor / 2 \leq n/10 \) medians \( \leq p \).
Analysis of median-of-medians selection algorithm

- At least half of 5-element medians $\leq p$.
- At least $\lfloor n/5 \rfloor / 2 = \lfloor n/10 \rfloor$ medians $\leq p$.
- At least 3 $\lfloor n/10 \rfloor$ elements $\leq p$. 

\[ \text{median of medians } p \]
Analysis of median-of-medians selection algorithm

- At least half of 5-element medians $\geq p$. 

N = 54
Analysis of median-of-medians selection algorithm

- At least half of 5-element medians $\geq p$.
- Symmetrically, at least $\lceil n / 10 \rceil$ medians $\geq p$. 

median of medians $p$

N = 54
Analysis of median-of-medians selection algorithm

- At least half of 5-element medians $\geq p$.
- Symmetrically, at least $\lceil n / 10 \rceil$ medians $\geq p$.
- At least 3 $\lfloor n / 10 \rfloor$ elements $\geq p$.  

$N = 54$
Median-of-medians selection algorithm recurrence

Median-of-medians selection algorithm recurrence.
- Select called recursively with $\lfloor n/5 \rfloor$ elements to compute MOM $p$.
- At least $3 \lfloor n/10 \rfloor$ elements $\leq p$.
- At least $3 \lfloor n/10 \rfloor$ elements $\geq p$.
- Select called recursively with at most $n - 3 \lfloor n/10 \rfloor$ elements.

Def. $C(n) = \max \#$ compares on an array of $n$ elements.

$$C(n) \leq C(\lfloor n/5 \rfloor) + C(n - 3 \lfloor n/10 \rfloor) + \frac{11}{5} n$$

median of medians \hspace{1cm} \text{recursive select} \hspace{1cm} \text{computing median of 5 (6 compares per group)} \hspace{1cm} \text{partitioning (n compares)}$

Now, solve recurrence.
- Assume $n$ is both a power of 5 and a power of 10?
- Assume $C(n)$ is monotone nondecreasing?
Median-of-medians selection algorithm recurrence

Analysis of selection algorithm recurrence.

• \( T(n) = \text{max # compares on an array of } \leq n \text{ elements}. \)

• \( T(n) \) is monotone, but \( C(n) \) is not!

\[
T(n) \leq \begin{cases} 
6n & \text{if } n < 50 \\
T(\lfloor n/5 \rfloor) + T(n - 3 \lfloor n/10 \rfloor) + \frac{1}{5} n & \text{otherwise}
\end{cases}
\]

Claim. \( T(n) \leq 44n. \)

• Base case: \( T(n) \leq 6n \) for \( n < 50 \) (mergesort).

• Inductive hypothesis: assume true for \( 1, 2, \ldots, n-1. \)

• Induction step: for \( n \geq 50, \) we have:

\[
T(n) \leq T(\lfloor n / 5 \rfloor) + T(n - 3 \lfloor n / 10 \rfloor) + 11/5 n
\]
\[
\leq 44 (\lfloor n / 5 \rfloor) + 44 (n - 3 \lfloor n / 10 \rfloor) + 11/5 n
\]
\[
\leq 44 (n / 5) + 44 n - 44 (n / 4) + 11/5 n \quad \text{for } n \geq 50, \ 3 \lfloor n / 10 \rfloor \geq n / 4
\]
\[
= 44 n. \quad \blacksquare
\]
Linear-time selection postmortem

**Proposition.** [Blum-Floyd-Pratt-Rivest-Tarjan 1973] There exists a compare-based selection algorithm whose worst-case running time is $O(n)$.

---

**Time Bounds for Selection**

by .

Manuel Blum, Robert W. Floyd, Vaughan Pratt,
Ronald L. Rivest, and Robert E. Tarjan

**Abstract**

The number of comparisons required to select the $i$-th smallest of $n$ numbers is shown to be at most a linear function of $n$ by analysis of a new selection algorithm -- PICK. Specifically, no more than $5.4305n$ comparisons are ever required. This bound is improved for

---

**Theory.**

- Optimized version of BFPRT: $\leq 5.4305n$ compares.
- Best known upper bound [Dor-Zwick 1995]: $\leq 2.95n$ compares.
- Best known lower bound [Dor-Zwick 1999]: $\geq (2 + \varepsilon)n$ compares.
Linear-time selection postmortem

**Proposition.** [Blum-Floyd-Pratt-Rivest-Tarjan 1973] There exists a compare-based selection algorithm whose worst-case running time is $O(n)$.

**Abstract.**

The number of comparisons required to select the $i$-th smallest of $n$ numbers is shown to be at most a linear function of $n$ by analysis of a new selection algorithm -- PICK. Specifically, no more than $5.4305n$ comparisons are ever required. This bound is improved for extreme values of $i$.

**Practice.** Constant and overhead (currently) too large to be useful.

**Open.** Practical selection algorithm whose worst-case running time is $O(n)$. 
DIVIDE AND CONQUER II

- master theorem
- integer multiplication
- matrix multiplication
- convolution and FFT
Sections 4.3–4.6

Divide and Conquer II

- master theorem
- integer multiplication
- matrix multiplication
- convolution and FFT
**Master method**

**Goal.** Recipe for solving common divide-and-conquer recurrences:

\[ T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n) \]

**Terms.**
- \( a \geq 1 \) is the number of subproblems.
- \( b > 0 \) is the factor by which the subproblem size decreases.
- \( f(n) = \) work to divide/merge subproblems.

**Recursion tree.**
- \( k = \log_b n \) levels.
- \( a^i = \) number of subproblems at level \( i \).
- \( n / b^i = \) size of subproblem at level \( i \).
Case 1: total cost dominated by cost of leaves

Ex 1. If $T(n)$ satisfies $T(n) = 3 \ T(n/2) + n$, with $T(1) = 1$, then $T(n) = \Theta(n\log^3)$. 

$$r = \frac{3}{2} > 1 \quad T(n) = (1 + r + r^2 + r^3 + \ldots + r^{\log_2 n}) n = \frac{r^{1+\log_2 n} - 1}{r - 1} n = 3n^{\log_2 3} - 2n$$
Case 2: total cost evenly distributed among levels

Ex 2. If $T(n)$ satisfies $T(n) = 2 \, T(n/2) + n$, with $T(1) = 1$, then $T(n) = \Theta(n \log n)$. 

$$r = 1 \quad T(n) = (1 + r + r^2 + r^3 + \ldots + r^{\log_2 n}) \, n = n \, (\log_2 n + 1)$$
Case 3: total cost dominated by cost of root

Ex 3. If $T(n)$ satisfies $T(n) = 3\ T(n/4) + n^5$, with $T(1) = 1$, then $T(n) = \Theta(n^5)$.

![Diagram]

\[
r = \frac{3}{4^5} < 1 \quad n^5 \leq T(n) \leq (1 + r + r^2 + r^3 + \ldots) n^5 \leq \frac{1}{1 - r} \ n^5
\]
**Master theorem**

**Master theorem.** Suppose that $T(n)$ is a function on the nonnegative integers that satisfies the recurrence

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

where $n/b$ means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Let $k = \log_b a$. Then,

**Case 1.** If $f(n) = O(n^{k-\varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^k)$.

**Ex.** $T(n) = 3 \cdot T(n/2) + n$.

- $a = 3$, $b = 2$, $f(n) = n$, $k = \log_2 3$.
- $T(n) = \Theta(n \lg^3 3)$.
**Master theorem**

**Master theorem.** Suppose that $T(n)$ is a function on the nonnegative integers that satisfies the recurrence

$$T(n) = a \, T\left(\frac{n}{b}\right) + f(n)$$

where $n/b$ means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Let $k = \log_b a$. Then,

**Case 2.** If $f(n) = \Theta(n^k \log^p n)$, then $T(n) = \Theta(n^k \log^{p+1} n)$.

**Ex.** $T(n) = 2 \, T(n/2) + \Theta(n \log n)$.

- $a = 2, \ b = 2, \ f(n) = 17n, \ k = \log_2 2 = 1, \ p = 1$.
- $T(n) = \Theta(n \log^2 n)$. 
Master theorem

Master theorem. Suppose that \( T(n) \) is a function on the nonnegative integers that satisfies the recurrence

\[
T(n) = a \ T\left(\frac{n}{b}\right) + f(n)
\]

where \( n/b \) means either \( \lfloor n/b \rfloor \) or \( \lceil n/b \rceil \). Let \( k = \log_b a \). Then,

Case 3. If \( f(n) = \Omega(n^k + \varepsilon) \) for some constant \( \varepsilon > 0 \) and if \( af(n/b) \leq cf(n) \) for some constant \( c < 1 \) and all sufficiently large \( n \), then \( T(n) = \Theta(f(n)) \).

Ex. \( T(n) = 3 \ T(n / 4) + n^5 \).

- \( a = 3, \ b = 4, \ f(n) = n^5, \ k = \log_4 3 \).
- \( T(n) = \Theta(n^5) \).
Master theorem

Master theorem. Suppose that $T(n)$ is a function on the nonnegative integers that satisfies the recurrence

$$T(n) = a \cdot T \left( \frac{n}{b} \right) + f(n)$$

where $n/b$ means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Let $k = \log_b a$. Then,

Case 1. If $f(n) = O(n^{k-\epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^k)$.

Case 2. If $f(n) = \Theta(n^k \log^p n)$, then $T(n) = \Theta(n^k \log^{p+1} n)$.

Case 3. If $f(n) = \Omega(n^k + \epsilon)$ for some constant $\epsilon > 0$ and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large $n$, then $T(n) = \Theta(f(n))$.

Pf sketch.

- Use recursion tree to sum up terms (assuming $n$ is an exact power of $b$).
- Three cases for geometric series.
- Deal with floors and ceilings.
Akra-Bazzi theorem

Desiderata. Generalizes master theorem to divide-and-conquer algorithms where subproblems have substantially different sizes.

Theorem. [Akra-Bazzi] Given constants $a_i > 0$ and $0 < b_i \leq 1$, functions $h_i(n) = O(n / \log^2 n)$ and $g(n) = O(n^c)$, if the function $T(n)$ satisfies the recurrence:

$$T(n) = \sum_{i=1}^{k} a_i T(b_i n + h_i(n)) + g(n)$$

Then $T(n) = \Theta \left( n^p \left( 1 + \int_{1}^{n} \frac{g(u)}{u^{p+1}} du \right) \right)$ where $p$ satisfies $\sum_{i=1}^{k} a_i b_i^p = 1$.

Ex. $T(n) = 7/4 \ T(\lfloor n/2 \rfloor) + T(\lceil 3/4 \ n \rceil) + n^2$.

- $a_1 = 7/4, \ b_1 = 1/2, \ a_2 = 1, \ b_2 = 3/4 \Rightarrow \ p = 2$.
- $h_1(n) = \lfloor 1/2 \ n \rfloor - 1/2 \ n, \ h_2(n) = \lceil 3/4 \ n \rceil - 3/4 \ n$.
- $g(n) = n^2 \Rightarrow T(n) = \Theta(n^2 \log n)$. 
Section 5.5

Divide and Conquer II

- master theorem
- integer multiplication
- matrix multiplication
- convolution and FFT
Integer addition

**Addition.** Given two $n$-bit integers $a$ and $b$, compute $a + b$.

**Subtraction.** Given two $n$-bit integers $a$ and $b$, compute $a - b$.

**Grade-school algorithm.** $\Theta(n)$ bit operations.

![Addition Example](image.png)

**Remark.** Grade-school addition and subtraction algorithms are asymptotically optimal.
**Integer multiplication**

**Multiplication.** Given two $n$-bit integers $a$ and $b$, compute $a \times b$.

**Grade-school algorithm.** $\Theta(n^2)$ bit operations.

![Integer multiplication diagram]

**Conjecture.** [Kolmogorov 1952] Grade-school algorithm is optimal.

**Theorem.** [Karatsuba 1960] Conjecture is wrong.
Divide-and-conquer multiplication

To multiply two $n$-bit integers $x$ and $y$:

- Divide $x$ and $y$ into low- and high-order bits.
- Multiply four $\frac{1}{2}n$-bit integers, recursively.
- Add and shift to obtain result.

$$m = \left\lfloor \frac{n}{2} \right\rfloor$$

$$a = \left\lfloor \frac{x}{2^m} \right\rfloor \quad \quad \quad b = x \mod 2^m$$

$$c = \left\lfloor \frac{y}{2^m} \right\rfloor \quad \quad \quad d = y \mod 2^m$$

$$(2^m a + b) (2^m c + d) = 2^{2m} ac + 2^m (bc + ad) + bd$$

Ex. $x = 10001101$  $y = 11100001$

\[\begin{array}{c}
a \\
b \\
c \\
d
\end{array}\]
Divide-and-conquer multiplication

\[ \text{MULTIPLY}(x, y, n) \]

\[
\begin{align*}
\text{IF} & \quad (n = 1) \\
& \quad \text{RETURN } x \times y. \\
\text{ELSE} & \\
& \quad m \leftarrow \lfloor n / 2 \rfloor. \\
& \quad a \leftarrow \lfloor x / 2^m \rfloor; \quad b \leftarrow x \mod 2^m. \\
& \quad c \leftarrow \lfloor y / 2^m \rfloor; \quad d \leftarrow y \mod 2^m. \\
& \quad e \leftarrow \text{MULTIPLY}(a, c, m). \\
& \quad f \leftarrow \text{MULTIPLY}(b, d, m). \\
& \quad g \leftarrow \text{MULTIPLY}(b, c, m). \\
& \quad h \leftarrow \text{MULTIPLY}(a, d, m). \\
& \quad \text{RETURN } 2^m e + 2^m (g + h) + f.
\end{align*}
\]
Divide-and-conquer multiplication analysis

Proposition. The divide-and-conquer multiplication algorithm requires $\Theta(n^2)$ bit operations to multiply two $n$-bit integers.

Pf. Apply case 1 of the master theorem to the recurrence:

$$T(n) = 4T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n^2)$$
Karatsuba trick

To compute middle term $bc + ad$, use identity:

$$bc + ad = ac + bd - (a - b)(c - d)$$

$$m = \lfloor n / 2 \rfloor$$

$$a = \lfloor x / 2^m \rfloor \quad b = x \mod 2^m$$

$$c = \lfloor y / 2^m \rfloor \quad d = y \mod 2^m$$

$$(2^m a + b)(2^m c + d) = 2^{2m} ac + 2^m (bc + ad) + bd$$

$$= 2^{2m} ac + 2^m (ac + bd - (a - b)(c - d)) + bd$$

Bottom line. Only three multiplication of $n/2$-bit integers.
Karatsuba multiplication

**Karatsuba-Multiply**$(x, y, n)$

**IF** $(n = 1)$

**RETURN** $x \times y$.

**ELSE**

$m \leftarrow \lfloor n / 2 \rfloor$.

$a \leftarrow \lfloor x / 2^m \rfloor$; \hspace{0.5cm} $b \leftarrow x \bmod 2^m$.

$c \leftarrow \lfloor y / 2^m \rfloor$; \hspace{0.5cm} $d \leftarrow y \bmod 2^m$.

$e \leftarrow \text{Karatsuba-Multiply}(a, c, m)$.

$f \leftarrow \text{Karatsuba-Multiply}(b, d, m)$.

$g \leftarrow \text{Karatsuba-Multiply}(a - b, c - d, m)$.

**RETURN** $2^m e + 2^m (e + f - g) + f$. 
Karatsuba analysis

**Proposition.** Karatsuba's algorithm requires $O(n^{1.585})$ bit operations to multiply two $n$-bit integers.

**Pf.** Apply case 1 of the master theorem to the recurrence:

$$T(n) = 3 T(n/2) + \Theta(n) \implies T(n) = \Theta(n^{\lg 3}) = O(n^{1.585}).$$

**Practice.** Faster than grade-school algorithm for about 320-640 bits.
Integer arithmetic reductions

**Integer multiplication.** Given two $n$-bit integers, compute their product.

<table>
<thead>
<tr>
<th>problem</th>
<th>arithmetic</th>
<th>running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer multiplication</td>
<td>$a \times b$</td>
<td>$\Theta(M(n))$</td>
</tr>
<tr>
<td>integer division</td>
<td>$a / b$, $a \mod b$</td>
<td>$\Theta(M(n))$</td>
</tr>
<tr>
<td>integer square</td>
<td>$a^2$</td>
<td>$\Theta(M(n))$</td>
</tr>
<tr>
<td>integer square root</td>
<td>$\lfloor \sqrt{a} \rfloor$</td>
<td>$\Theta(M(n))$</td>
</tr>
</tbody>
</table>

*integer arithmetic problems with the same complexity as integer multiplication*
## History of asymptotic complexity of integer multiplication

<table>
<thead>
<tr>
<th>Year</th>
<th>Algorithm</th>
<th>Order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>brute force</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>1962</td>
<td>Karatsuba-Ofman</td>
<td>$\Theta(n^{1.585})$</td>
</tr>
<tr>
<td>1963</td>
<td>Toom-3, Toom-4</td>
<td>$\Theta(n^{1.465}), \Theta(n^{1.404})$</td>
</tr>
<tr>
<td>1966</td>
<td>Toom-Cook</td>
<td>$\Theta(n^{1+\varepsilon})$</td>
</tr>
<tr>
<td>1971</td>
<td>Schönhage–Strassen</td>
<td>$\Theta(n \log n \log \log n)$</td>
</tr>
<tr>
<td>2007</td>
<td>Fürer</td>
<td>$n \log n \ 2^{O(\log^* n)}$</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>$\Theta(n)$</td>
</tr>
</tbody>
</table>

**number of bit operations to multiply two $n$-bit integers**

used in Maple, Mathematica, gcc, cryptography, ...

**Remark.** GNU Multiple Precision Library uses one of five different algorithm depending on size of operands.
Dot product

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

Grade-school. \( \Theta(n) \) arithmetic operations.

\[
a \cdot b = \sum_{i=1}^{n} a_i b_i
\]

\[
a = \begin{bmatrix} .70 & .20 & .10 \end{bmatrix}
\]
\[
b = \begin{bmatrix} .30 & .40 & .30 \end{bmatrix}
\]

\[
a \cdot b = (.70 \times .30) + (.20 \times .40) + (.10 \times .30) = .32
\]

Remark. Grade-school dot product algorithm is asymptotically optimal.
Matrix multiplication

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

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

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

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

Q. Is grade-school matrix multiplication algorithm asymptotically optimal?
Block matrix multiplication

\[
C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}
\]

\[
= \begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix}
\]

\[
= \begin{bmatrix} 152 & 158 \\ 504 & 526 \end{bmatrix}
\]
Matrix multiplication: warmup

To multiply two $n$-by-$n$ matrices $A$ and $B$:

- **Divide:** partition $A$ and $B$ into \( \frac{1}{2}n \)-by-\( \frac{1}{2}n \) blocks.
- **Conquer:** multiply 8 pairs of \( \frac{1}{2}n \)-by-\( \frac{1}{2}n \) matrices, recursively.
- **Combine:** add appropriate products using 4 matrix additions.

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

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

**Running time.** Apply case 1 of Master Theorem.

\[
T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2) \quad \Rightarrow \quad T(n) = \Theta(n^3)
\]
Strassen's trick

**Key idea.** multiply 2-by-2 blocks with only 7 multiplications. (plus 11 additions and 7 subtractions)

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

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

**Pf.** \(C_{12} = P_1 + P_2\)
\[
= A_{11} \times (B_{12} - B_{22}) + (A_{11} + A_{12}) \times B_{22}
\]
\[
= A_{11} \times B_{12} + A_{12} \times B_{22}. \quad \checkmark
\]
Strassen's algorithm

**Strassen** \((n, A, B)\)

**IF** \((n = 1)\) **RETURN** \(A \times B\).

Partition \(A\) and \(B\) into 2-by-2 block matrices.

- \(P_1 \leftarrow \text{Strassen}(n/2, A_{11}, (B_{12} - B_{22}))\).
- \(P_2 \leftarrow \text{Strassen}(n/2, (A_{11} + A_{12}), B_{22})\).
- \(P_3 \leftarrow \text{Strassen}(n/2, (A_{21} + A_{22}), B_{11})\).
- \(P_4 \leftarrow \text{Strassen}(n/2, A_{22}, (B_{21} - B_{11}))\).
- \(P_5 \leftarrow \text{Strassen}(n/2, (A_{11} + A_{22}) \times (B_{11} + B_{22}))\).
- \(P_6 \leftarrow \text{Strassen}(n/2, (A_{12} - A_{22}) \times (B_{21} + B_{22}))\).
- \(P_7 \leftarrow \text{Strassen}(n/2, (A_{11} - A_{21}) \times (B_{11} + B_{12}))\).

\[
\begin{align*}
C_{11} &= P_5 + P_4 - P_2 + P_6. \\
C_{12} &= P_1 + P_2. \\
C_{21} &= P_3 + P_4. \\
C_{22} &= P_1 + P_5 - P_3 - P_7. \\
\end{align*}
\]

**RETURN** \(C\).
Analysis of Strassen's algorithm

**Theorem.** Strassen's algorithm requires $O(n^{2.81})$ arithmetic operations to multiply two $n$-by-$n$ matrices.

**Pf.** Apply case 1 of the master theorem to the recurrence:

$$T(n) = 7T(n/2) + \Theta(n^2) \quad \Rightarrow \quad T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

**Q.** What if $n$ is not a power of 2?

**A.** Could pad matrices with zeros.

\[
\begin{bmatrix}
1 & 2 & 3 & 0 \\
4 & 5 & 6 & 0 \\
7 & 8 & 9 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\times
\begin{bmatrix}
10 & 11 & 12 & 0 \\
13 & 14 & 15 & 0 \\
16 & 17 & 18 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
= 
\begin{bmatrix}
84 & 90 & 96 & 0 \\
201 & 216 & 231 & 0 \\
318 & 342 & 366 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]
Strassen's algorithm: practice

Implementation issues.
• Sparsity.
• Caching effects.
• Numerical stability.
• Odd matrix dimensions.
• Crossover to classical algorithm when $n$ is "small".

Common misperception. “Strassen is only a theoretical curiosity.”
• Apple reports 8x speedup on G4 Velocity Engine when $n \approx 2,048$.
• Range of instances where it's useful is a subject of controversy.
Linear algebra reductions

Matrix multiplication. Given two \( n \)-by-\( n \) matrices, compute their product.

<table>
<thead>
<tr>
<th>problem</th>
<th>linear algebra</th>
<th>order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix multiplication</td>
<td>( A \times B )</td>
<td>( \Theta(MM(n)) )</td>
</tr>
<tr>
<td>matrix inversion</td>
<td>( A^{-1} )</td>
<td>( \Theta(MM(n)) )</td>
</tr>
<tr>
<td>determinant</td>
<td>(</td>
<td>A</td>
</tr>
<tr>
<td>system of linear equations</td>
<td>( Ax = b )</td>
<td>( \Theta(MM(n)) )</td>
</tr>
<tr>
<td>LU decomposition</td>
<td>( A = LU )</td>
<td>( \Theta(MM(n)) )</td>
</tr>
<tr>
<td>least squares</td>
<td>min ( |Ax - b|_2 )</td>
<td>( \Theta(MM(n)) )</td>
</tr>
</tbody>
</table>

Numerical linear algebra problems with the same complexity as matrix multiplication.
Fast matrix multiplication: theory

Q. Multiply two 2-by-2 matrices with 7 scalar multiplications?
A. Yes! [Strassen 1969]
\[ \Theta(n^{\log_2 7}) = O(n^{2.807}) \]

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

Q. Multiply two 3-by-3 matrices with 21 scalar multiplications?
A. Unknown.
\[ \Theta(n^{\log_3 21}) = O(n^{2.77}) \]

Begun, the decimal wars have. [Pan, Bini et al, Schönhage, ...]
• Two 20-by-20 matrices with 4,460 scalar multiplications. \( O(n^{2.805}) \)
• Two 48-by-48 matrices with 47,217 scalar multiplications. \( O(n^{2.7801}) \)
• A year later.
• December 1979.
• January 1980.
# History of asymptotic complexity of matrix multiplication

<table>
<thead>
<tr>
<th>year</th>
<th>algorithm</th>
<th>order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>brute force</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>1969</td>
<td>Strassen</td>
<td>$O(n^{2.808})$</td>
</tr>
<tr>
<td>1978</td>
<td>Pan</td>
<td>$O(n^{2.796})$</td>
</tr>
<tr>
<td>1979</td>
<td>Bini</td>
<td>$O(n^{2.780})$</td>
</tr>
<tr>
<td>1981</td>
<td>Schönhage</td>
<td>$O(n^{2.522})$</td>
</tr>
<tr>
<td>1982</td>
<td>Romani</td>
<td>$O(n^{2.517})$</td>
</tr>
<tr>
<td>1982</td>
<td>Coppersmith-Winograd</td>
<td>$O(n^{2.496})$</td>
</tr>
<tr>
<td>1986</td>
<td>Strassen</td>
<td>$O(n^{2.479})$</td>
</tr>
<tr>
<td>1989</td>
<td>Coppersmith-Winograd</td>
<td>$O(n^{2.376})$</td>
</tr>
<tr>
<td>2010</td>
<td>Strother</td>
<td>$O(n^{2.3737})$</td>
</tr>
<tr>
<td>2011</td>
<td>Williams</td>
<td>$O(n^{2.3727})$</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>$O(n^2 + \varepsilon)$</td>
</tr>
</tbody>
</table>

number of floating-point operations to multiply two $n$-by-$n$ matrices
Section 5.6

Divide and Conquer II

- master theorem
- integer multiplication
- matrix multiplication
- convolution and FFT
Fourier analysis

**Fourier theorem.** [Fourier, Dirichlet, Riemann] Any (sufficiently smooth) periodic function can be expressed as the sum of a series of sinusoids.

\[ y(t) = \frac{2}{\pi} \sum_{k=1}^{N} \frac{\sin kt}{k} \quad N = 100 \]
Euler's identity

Euler's identity. $e^{ix} = \cos x + i \sin x$.

Sinusoids. Sum of sine and cosines = sum of complex exponentials.
Time domain vs. frequency domain

**Signal.** [touch tone button 1] \[ y(t) = \frac{1}{2} \sin(2\pi \cdot 697 \cdot t) + \frac{1}{2} \sin(2\pi \cdot 1209 \cdot t) \]

Time domain.  

Frequency domain.  

Reference: Cleve Moler, Numerical Computing with MATLAB
**Time domain vs. frequency domain**

**Signal.** [recording, 8192 samples per second]

Magnitude of discrete Fourier transform.

Reference: Cleve Moler, Numerical Computing with MATLAB
Fast Fourier transform

**FFT.** Fast way to convert between time-domain and frequency-domain.

**Alternate viewpoint.** Fast way to multiply and evaluate polynomials.

“If you speed up any nontrivial algorithm by a factor of a million or so the world will beat a path towards finding useful applications for it.” — Numerical Recipes
Fast Fourier transform: applications

Applications.

- Optics, acoustics, quantum physics, telecommunications, radar, control systems, signal processing, speech recognition, data compression, image processing, seismology, mass spectrometry, ...
- Digital media. [DVD, JPEG, MP3, H.264]
- Medical diagnostics. [MRI, CT, PET scans, ultrasound]
- Numerical solutions to Poisson's equation.
- Integer and polynomial multiplication.
- Shor's quantum factoring algorithm.
- ...

“The FFT is one of the truly great computational developments of [the 20th] century. It has changed the face of science and engineering so much that it is not an exaggeration to say that life as we know it would be very different without the FFT.”

— Charles van Loan
Fast Fourier transform: brief history

**Gauss (1805, 1866).** Analyzed periodic motion of asteroid Ceres.

**Runge-König (1924).** Laid theoretical groundwork.

**Danielson-Lanczos (1942).** Efficient algorithm, x-ray crystallography.

**Cooley-Tukey (1965).** Monitoring nuclear tests in Soviet Union and tracking submarines. Rediscovered and popularized FFT.

---

**An Algorithm for the Machine Calculation of Complex Fourier Series**

By James W. Cooley and John W. Tukey

An efficient method for the calculation of the interactions of a $2^n$ factorial experiment was introduced by Yates and is widely known by his name. The generalization to $3^n$ was given by Box et al. [1]. Good [2] generalized these methods and gave elegant algorithms for which one class of applications is the calculation of Fourier series. In their full generality, Good’s methods are applicable to certain problems in which one must multiply an $N$-vector by an $N \times N$ matrix which can be factored into $m$ sparse matrices, where $m$ is proportional to $\log N$. This results in a procedure requiring a number of operations proportional to $N \log N$ rather than $N^2$.

---

Paper published only after IBM lawyers decided not to set precedent of patenting numerical algorithms
(conventional wisdom: nobody could make money selling software!)

**Importance** not fully realized until advent of digital computers.
Polynomials: coefficient representation

**Polynomial.** [coefficient representation]

\[ A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} \]

\[ B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1} \]

**Add.** \(O(n)\) arithmetic operations.

\[ A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1) x + \cdots + (a_{n-1} + b_{n-1}) x^{n-1} \]

**Evaluate.** \(O(n)\) using Horner's method.

\[ A(x) = a_0 + (x(a_1 + x(a_2 + \cdots + x(a_{n-2} + x(a_{n-1})))) \cdots) \]

**Multiply (convolve).** \(O(n^2)\) using brute force.

\[ A(x) \times B(x) = \sum_{i=0}^{2n-2} c_i x^i, \text{ where } c_i = \sum_{j=0}^{i} a_j b_{i-j} \]
"New proof of the theorem that every algebraic rational integral function in one variable can be resolved into real factors of the first or the second degree."
Polynomials: point-value representation

**Fundamental theorem of algebra.** A degree $n$ polynomial with complex coefficients has exactly $n$ complex roots.

**Corollary.** A degree $n-1$ polynomial $A(x)$ is uniquely specified by its evaluation at $n$ distinct values of $x$. 

![Graph showing polynomial function with points and axis labels]
Polynomials: point-value representation

**Polynomial.** [point-value representation]

\[
A(x) : (x_0, y_0), \ldots, (x_{n-1}, y_{n-1}) \\
B(x) : (x_0, z_0), \ldots, (x_{n-1}, z_{n-1})
\]

**Add.** \(O(n)\) arithmetic operations.

\[
A(x) + B(x) : (x_0, y_0 + z_0), \ldots, (x_{n-1}, y_{n-1} + z_{n-1})
\]

**Multiply (convolve).** \(O(n)\), but need \(2n - 1\) points.

\[
A(x) \times B(x) : (x_0, y_0 \times z_0), \ldots, (x_{2n-1}, y_{2n-1} \times z_{2n-1})
\]

**Evaluate.** \(O(n^2)\) using Lagrange's formula.

\[
A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)}
\]
Converting between two representations

**Tradeoff.** Fast evaluation or fast multiplication. We want both!

<table>
<thead>
<tr>
<th>representation</th>
<th>multiply</th>
<th>evaluate</th>
</tr>
</thead>
<tbody>
<tr>
<td>coefficient</td>
<td>$O(n^2)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>point-value</td>
<td>$O(n)$</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>

**Goal.** Efficient conversion between two representations $\Rightarrow$ all ops fast.
Converting between two representations: brute force

**Coefficient ⇒ point-value.** Given a polynomial \( a_0 + a_1 x + ... + a_{n-1} x^{n-1} \), evaluate it at \( n \) distinct points \( x_0, ..., x_{n-1} \).

\[
\begin{bmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_{n-1}
\end{bmatrix} =
\begin{bmatrix}
1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\
1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\
1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1}
\end{bmatrix}
\begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_{n-1}
\end{bmatrix}
\]

**Running time.** \( O(n^2) \) for matrix-vector multiply (or \( n \) Horner's).
Converting between two representations: brute force

**Point-value ⇒ coefficient.** Given $n$ distinct points $x_0, \ldots, x_{n-1}$ and values $y_0, \ldots, y_{n-1}$, find unique polynomial $a_0 + a_1x + \ldots + a_{n-1}x^{n-1}$, that has given values at given points.

\[
\begin{bmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_{n-1}
\end{bmatrix} =
\begin{bmatrix}
1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\
1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\
1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1}
\end{bmatrix}
\begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_{n-1}
\end{bmatrix}
\]

Vandermonde matrix is invertible iff $x_i$ distinct

**Running time.** $O(n^3)$ for Gaussian elimination.

or $O(n^{2.3727})$ via fast matrix multiplication
Divide-and-conquer

Decimation in frequency. Break up polynomial into low and high powers.

- \( A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7. \)
- \( A_{low}(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3. \)
- \( A_{high}(x) = a_4 + a_5 x + a_6 x^2 + a_7 x^3. \)
- \( A(x) = A_{low}(x) + x^4 A_{high}(x). \)

Decimation in time. Break up polynomial into even and odd powers.

- \( A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7. \)
- \( A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3. \)
- \( A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3. \)
- \( A(x) = A_{even}(x^2) + x A_{odd}(x^2). \)
Coefficient to point-value representation: intuition

**Coefficient ⇒ point-value.** Given a polynomial \( a_0 + a_1 x + \ldots + a_{n-1} x^{n-1} \), evaluate it at \( n \) distinct points \( x_0, \ldots, x_{n-1} \). (we get to choose which ones!)

**Divide.** Break up polynomial into even and odd powers.

- \( A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7 \).
- \( A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3 \).
- \( A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3 \).
- \( A(x) = A_{even}(x^2) + x A_{odd}(x^2) \).
- \( A(-x) = A_{even}(x^2) - x A_{odd}(x^2) \).

**Intuition.** Choose two points to be \( \pm 1 \).

- \( A(1) = A_{even}(1) + 1 A_{odd}(1) \).
- \( A(-1) = A_{even}(1) - 1 A_{odd}(1) \). (Can evaluate polynomial of degree \( \leq n \) at 2 points by evaluating two polynomials of degree \( \leq \frac{1}{2} n \) at 1 point.)
Coefficient to point-value representation: intuition

Coefficient $\Rightarrow$ point-value. Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at $n$ distinct points $x_0, \ldots, x_{n-1}$.

Divide. Break up polynomial into even and odd powers.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7$.
- $A_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3$.
- $A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3$.
- $A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2)$.
- $A(-x) = A_{\text{even}}(x^2) - x A_{\text{odd}}(x^2)$.

Intuition. Choose four complex points to be $\pm 1, \pm i$.

- $A(1) = A_{\text{even}}(1) + 1 A_{\text{odd}}(1)$.
- $A(-1) = A_{\text{even}}(1) - 1 A_{\text{odd}}(1)$.
- $A(i) = A_{\text{even}}(-1) + i A_{\text{odd}}(-1)$.
- $A(-i) = A_{\text{even}}(-1) - i A_{\text{odd}}(-1)$.

Can evaluate polynomial of degree $\leq n$ at 4 points by evaluating two polynomials of degree $\leq \frac{1}{2}n$ at 2 points.
Discrete Fourier transform

**Coefficient ⇒ point-value.** Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at $n$ distinct points $x_0, \ldots, x_{n-1}$.

**Key idea.** Choose $x_k = \omega^k$ where $\omega$ is principal $n^{th}$ root of unity.
Roots of unity

**Def.** An *n*\textsuperscript{th} root of unity is a complex number \(x\) such that \(x^n = 1\).

**Fact.** The *n*\textsuperscript{th} roots of unity are: \(\omega^0, \omega^1, \ldots, \omega^{n-1}\) where \(\omega = e^{2\pi i / n}\).

**Pf.** \((\omega^k)^n = (e^{2\pi i k / n})^n = (e^{\pi i})^{2k} = (-1)^{2k} = 1\).

**Fact.** The \(\frac{1}{2}n\textsuperscript{th}\) roots of unity are: \(\nu^0, \nu^1, \ldots, \nu^{n/2-1}\) where \(\nu = \omega^2 = e^{4\pi i / n}\).
Fast Fourier transform

**Goal.** Evaluate a degree \( n - 1 \) polynomial \( A(x) = a_0 + ... + a_{n-1} x^{n-1} \) at its \( n^{th} \) roots of unity: \( \omega^0, \omega^1, ..., \omega^{n-1} \).

**Divide.** Break up polynomial into even and odd powers.

- \( A_{even}(x) = a_0 + a_2 x + a_4 x^2 + ... + a_{n-2} x^{n/2-1} \).
- \( A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + ... + a_{n-1} x^{n/2-1} \).
- \( A(x) = A_{even}(x^2) + x A_{odd}(x^2) \).

**Conquer.** Evaluate \( A_{even}(x) \) and \( A_{odd}(x) \) at the \( \frac{1}{2}n^{th} \) roots of unity: \( \nu^0, \nu^1, ..., \nu^{n/2-1} \).

**Combine.**

- \( A(\omega^k) = A_{even}(\nu^k) + \omega^k A_{odd}(\nu^k), \ 0 \leq k < n/2 \)
- \( A(\omega^k + \frac{n}{2}) = A_{even}(\nu^k) - \omega^k A_{odd}(\nu^k), \ 0 \leq k < n/2 \)

\( \nu^k = (\omega^k)^2 \)

\( \nu^k = (\omega^k + \frac{n}{2})^2 \)

\( \omega^k + \frac{n}{2} = -\omega^k \)
**FFT: implementation**

\[
\text{FFT} (n, a_0, a_1, a_2, \ldots, a_{n-1})
\]

**IF** \((n = 1)\) **RETURN** \(a_0\).

\((e_0, e_1, \ldots, e_{n/2-1}) \leftarrow \text{FFT} (n/2, a_0, a_2, a_4, \ldots, a_{n-2}).\)

\((d_0, d_1, \ldots, d_{n/2-1}) \leftarrow \text{FFT} (n/2, a_1, a_3, a_4, \ldots, a_{n-1}).\)

**FOR** \(k = 0\) **TO** \(n/2 - 1.\)

\[\omega^k \leftarrow e^{2\pi ik/n}.\]

\[y_k \leftarrow e_k + \omega^k d_k.\]

\[y_{k+n/2} \leftarrow e_k - \omega^k d_k.\]

**RETURN** \((y_0, y_1, y_2, \ldots, y_{n-1}).\)
Theorem. The FFT algorithm evaluates a degree $n - 1$ polynomial at each of the $n^{th}$ roots of unity in $O(n \log n)$ steps and $O(n)$ extra space.

Pf. $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$

assumes $n$ is a power of 2

$\begin{align*}
\text{coefficient representation} & \quad O(n \log n) \quad \text{point-value representation} \\
(\alpha_0, \alpha_1, \ldots, \alpha_{n-1}) & \quad (x_0, y_0), \ldots, (x_{n-1}, y_{n-1})
\end{align*}$
FFT: recursion tree

- **Inverse perfect shuffle**

```
                        a₀, a₁, a₂, a₃, a₄, a₅, a₆, a₇
                       /   \
                   a₀, a₂, a₄, a₆   a₁, a₃, a₅, a₇
                  /   \         /   \
             a₀, a₄   a₂, a₆   a₁, a₅   a₃, a₇
            /   \   /   \   /   \   /   \
       a₀    a₄  a₂    a₆  a₁    a₅  a₃    a₇
```

"bit-reversed" order
Inverse discrete Fourier transform

**Point-value ⇒ coefficient.** Given $n$ distinct points $x_0, \ldots, x_{n-1}$ and values $y_0, \ldots, y_{n-1}$, find unique polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, that has given values at given points.

$$
\begin{bmatrix}
    a_0 \\
    a_1 \\
    a_2 \\
    a_3 \\
    \vdots \\
    a_{n-1}
\end{bmatrix} = \begin{bmatrix}
    1 & 1 & 1 & 1 & \cdots & 1 \\
    1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\
    1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\
    1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\
    \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
    1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)}
\end{bmatrix}^{-1} \begin{bmatrix}
    y_0 \\
    y_1 \\
    y_2 \\
    y_3 \\
    \vdots \\
    y_{n-1}
\end{bmatrix}
$$
Inverse discrete Fourier transform

Claim. Inverse of Fourier matrix $F_n$ is given by following formula:

\[
G_n = \frac{1}{n} \begin{bmatrix}
1 & 1 & 1 & 1 & \ldots & 1 \\
1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \ldots & \omega^{-(n-1)} \\
1 & \omega^{-2} & \omega^{-4} & \omega^{-6} & \ldots & \omega^{-2(n-1)} \\
1 & \omega^{-3} & \omega^{-6} & \omega^{-9} & \ldots & \omega^{-3(n-1)} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \omega^{-3(n-1)} & \ldots & \omega^{-(n-1)(n-1)}
\end{bmatrix}
\]

Consequence. To compute inverse FFT, apply same algorithm but use \(\omega^{-1} = e^{-2\pi i/n}\) as principal \(n^{th}\) root of unity (and divide the result by \(n\)).
Inverse FFT: proof of correctness

Claim. $F_n$ and $G_n$ are inverses.

Pf.

\[
\left( F_n \ G_n \right)_{kk'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{kj} \omega^{-jk'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{(k-k')j} = \begin{cases} 
1 & \text{if } k = k' \\
0 & \text{otherwise}
\end{cases}
\]

summation lemma (below)

Summation lemma. Let $\omega$ be a principal $n^{th}$ root of unity. Then

\[
\sum_{j=0}^{n-1} \omega^{kj} = \begin{cases} 
n & \text{if } k \equiv 0 \mod n \\
0 & \text{otherwise}
\end{cases}
\]

Pf.

- If $k$ is a multiple of $n$ then $\omega^k = 1 \Rightarrow$ series sums to $n$.
- Each $n^{th}$ root of unity $\omega^k$ is a root of $x^n - 1 = (x - 1)(1 + x + x^2 + \ldots + x^{n-1})$.
- If $\omega^k \neq 1$ we have: $1 + \omega^k + \omega^{k(2)} + \ldots + \omega^{k(n-1)} = 0 \Rightarrow$ series sums to 0. □
Inverse FFT: implementation

**Note.** Need to divide result by $n$.

\[\text{\textsc{Inverse-Fft}}(n, a_0, a_1, a_2, \ldots, a_{n-1})\]

**IF** $(n = 1)$ **RETURN** $a_0$.

$(e_0, e_1, \ldots, e_{n/2-1}) \leftarrow \text{\textsc{Inverse-Fft}}(n/2, a_0, a_2, a_4, \ldots, a_{n-2})$.

$(d_0, d_1, \ldots, d_{n/2-1}) \leftarrow \text{\textsc{Inverse-Fft}}(n/2, a_1, a_3, a_4, \ldots, a_{n-1})$.

**FOR** $k = 0$ **TO** $n/2 - 1$.

$\omega^k \leftarrow e^{-2\pi i k/n}$.

$y_k \leftarrow (e_k + \omega^k d_k)$.

$y_{k+n/2} \leftarrow (e_k - \omega^k d_k)$.

**RETURN** $(y_0, y_1, y_2, \ldots, y_{n-1})$.
**Inverse FFT: summary**

**Theorem.** The inverse FFT algorithm interpolates a degree \( n - 1 \) polynomial given values at each of the \( n^{th} \) roots of unity in \( O(n \log n) \) steps.

assumes \( n \) is a power of 2

---

\[ a_0, a_1, \ldots, a_{n-1} \]

**coefficient representation**

\[ (x_0, y_0), \ldots, (x_{n-1}, y_{n-1}) \]

**point-value representation**

\[ O(n \log n) \]

**(FFT)**

\[ O(n \log n) \]

**(inverse FFT)**
Polynomial multiplication

**Theorem.** Can multiply two degree \( n - 1 \) polynomials in \( O(n \log n) \) steps.

**Pf.**

- Pad with 0s to make \( n \) a power of 2

```
€
```

- \( a_0, a_1, \ldots, a_{n-1} \)
- \( b_0, b_1, \ldots, b_{n-1} \)

2 FFTs
\( O(n \log n) \)

- \( A(\omega^0), \ldots, A(\omega^{2n-1}) \)
- \( B(\omega^0), \ldots, B(\omega^{2n-1}) \)

point-value multiplication
\( O(n) \)

- \( C(\omega^0), \ldots, C(\omega^{2n-1}) \)

- \( c_0, c_1, \ldots, c_{2n-2} \)

- \( \omega_0, \ldots, \omega_{2n-1} \)

```
€
```
FFT in practice?
FFT in practice

Fastest Fourier transform in the West. [Frigo and Johnson]

- Optimized C library.
- Features: DFT, DCT, real, complex, any size, any dimension.
- Won 1999 Wilkinson Prize for Numerical Software.
- Portable, competitive with vendor-tuned code.

Implementation details.

- Core algorithm is nonrecursive version of Cooley-Tukey.
- Instead of executing predetermined algorithm, it evaluates your hardware and uses a special-purpose compiler to generate an optimized algorithm catered to "shape" of the problem.
- Runs in $O(n \log n)$ time, even when $n$ is prime.
- Multidimensional FFTs.

http://www.fftw.org
Integer multiplication, redux

**Integer multiplication.** Given two $n$-bit integers $a = a_{n-1} \ldots a_1 a_0$ and $b = b_{n-1} \ldots b_1 b_0$, compute their product $a \cdot b$.

**Convolution algorithm.**

- Form two polynomials. $A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$
- Note: $a = A(2)$, $b = B(2)$. $B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1}$
- Compute $C(x) = A(x) \cdot B(x)$.
- Evaluate $C(2) = a \cdot b$.
- Running time: $O(n \log n)$ complex arithmetic operations.

**Theory.** [Schönhage-Strassen 1971] $O(n \log n \log \log n)$ bit operations.

**Theory.** [Fürer 2007] $n \log n 2^{O(\log^* n)}$ bit operations.

**Practice.** [GNU Multiple Precision Arithmetic Library]

It uses brute force, Karatsuba, and FFT, depending on the size of $n$. 
