Chapter 5

Divide and Conquer

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 ½n.
- Solve two parts recursively.
- Combine two solutions into overall solution in linear time.

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

- # Mergesort

- # Counting Inversions

- # Integer Multiplication

- # Matrix Multiplication

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

Mergesort.

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

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

| A | L | G | O | R | | I | T | H | M | S | divide | $O(1)$ |

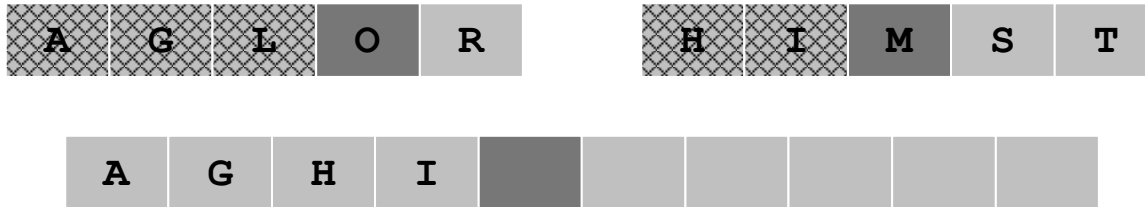| A | G | L | O | R | | H | I | M | S | T | sort | $2T(n/2)$ |

| A | G | H | I | L | M | O | R | S | T | merge | $O(n)$ |

Merging

Merging. Combine two pre-sorted lists into a sorted whole.

How to merge efficiently?
- Linear number of comparisons.
- Use temporary array.

A	G	L	O	R		H	I	M	S	T

A	G	H	I						

Challenge for the bored. In-place merge. [Kronrud, 1969]

↑

using only a constant amount of extra storage

A Useful Recurrence Relation

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

Mergesort recurrence.

$$T(n) \leq \begin{cases} 0 & if \ n = 1 \\ T\lceil n/2 \rceil + T\lceil n/2 \rceil + n & otherwise \end{cases}$$
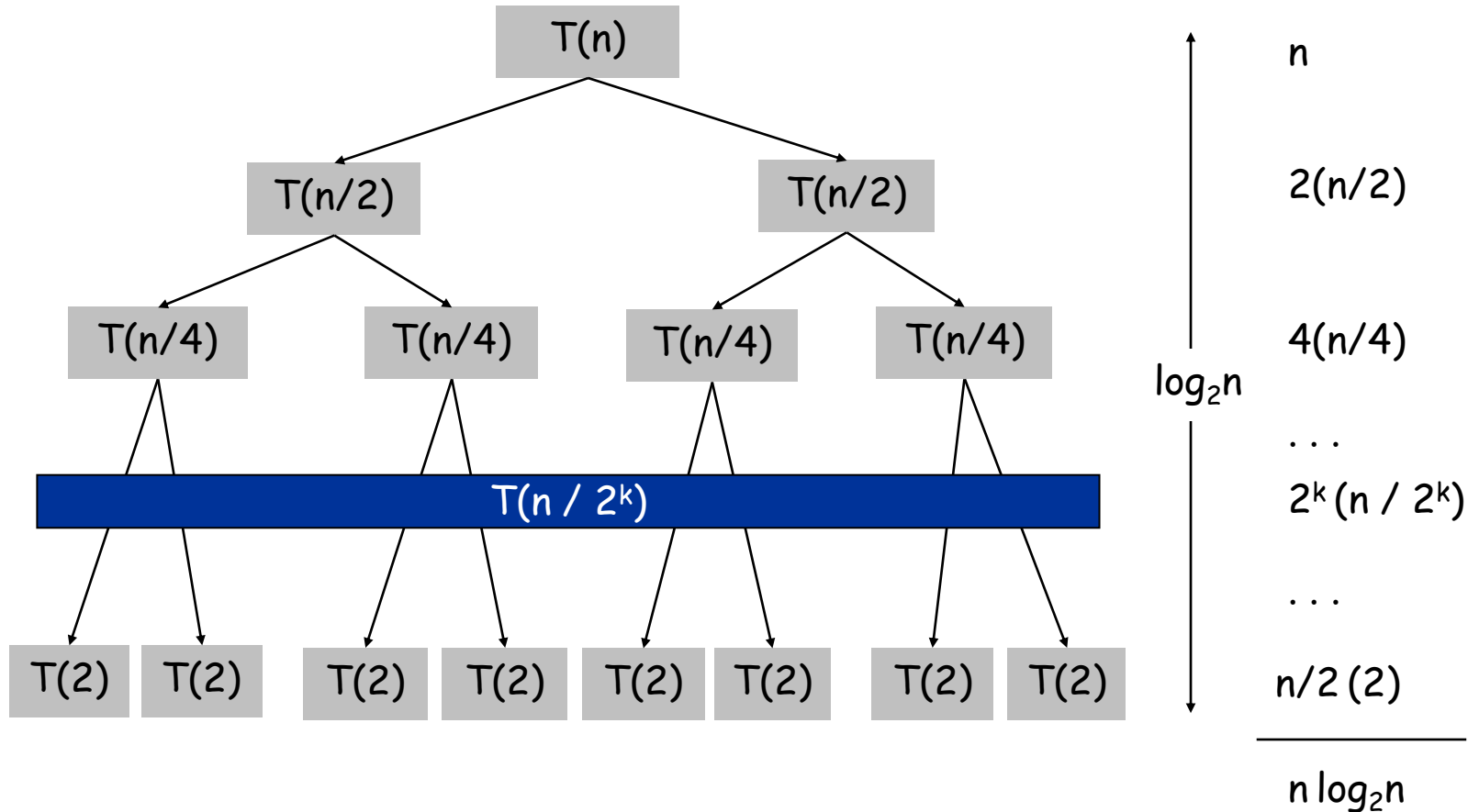
Solution. T(n) = O(n log$_2$ n).

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

Proof by Recursion Tree

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

T(n)	n
T(n/2) T(n/2)	2(n/2)
T(n/4) T(n/4) T(n/4) T(n/4)	4(n/4)
	. . .
T(n / 2^k)	$2^k (n / 2^k)$
	. . .
T(2) T(2) T(2) T(2) T(2) T(2) T(2) T(2)	n/2 (2)

$\log_2 n$

$n \log_2 n$

Proof by Induction

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

↑
assumes n is a power of 2

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

Pf. (by induction on n)

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

$$
\begin{aligned}
T(2n) &= 2T(n) + 2n \\
&= 2n \log_2 n + 2n \\
&= 2n\big(\log_2(2n) - 1\big) + 2n \\
&= 2n \log_2(2n)
\end{aligned}
$$

Master Theorem

Let T(n) be defined by the recurrence $T(n) = aT(n/b) + \Theta(n^d)$ for some constants $a \geq 1$, $b > 1$, and $d \geq 0$. Then we can conclude the following about the asymptotic compexity of T(n):

$$
T(n) = \begin{cases}
O(n^d) & si\ a < b^d \\
O(n^d \log n) & si\ a = b^d \\
O(n^{\log_b a}) & si\ a > b^d
\end{cases}
$$

5.3 Counting Inversions

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, …, n.
- Your rank: $a_1, a_2, …, a_n$.
- Songs i and j inverted if i < j, but $a_i > a_j$.

Songs

	A	B	C	D	E
Me	1	2	3	4	5
You	1	3	4	2	5

Inversions
3-2, 4-2

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

Applications

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).

Divide-and-conquer.

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

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.

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

Divide: O(1).

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

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

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

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

Divide: O(1).

1	5	4	8	10	2

6	9	12	11	3	7

Conquer: $2T(n / 2)$

5 blue-blue inversions

8 green-green inversions

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

6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half.
- Combine: count inversions where a_i and a_j are in different halves, and return sum of three quantities.

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

Divide: O(1).

| 1 | 5 | 4 | 8 | 10 | 2 |

5 blue-blue inversions

| 6 | 9 | 12 | 11 | 3 | 7 |

8 green-green inversions

Conquer: 2T(n / 2)

9 blue-green inversions
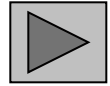5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

Combine: ???

$O(n^2)$

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.

to maintain sorted invariant

3	7	10	14	18	19

2	11	16	17	23	25
6	3	2	2	0	0

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

2	3	7	10	11	14	16	17	18	19	23	25

Merge: O(n)

$$T(n) \leq T\left(\lfloor n/2 \rfloor\right) + T\left(\lceil n/2 \rceil\right) + O(n) \;\Rightarrow\; \mathrm{T}(n) = O(n \log n)$$

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
}
```

5.5 Integer Multiplication

Integer Arithmetic

Add. Given two n-digit integers a and b, compute a + b.

- $O(n)$ bit operations.

Multiply. Given two n-digit integers a and b, compute a × b.

- Brute force solution: $\Theta(n^2)$ bit operations.

		1	1	0	1	0	1	0	1
	*	0	1	1	1	1	1	0	1
		1	1	0	1	0	1	0	1	0
		0	0	0	0	0	0	0	0	0
		1	1	0	1	0	1	0	1	0
		1	1	0	1	0	1	0	1	0
		1	1	0	1	0	1	0	1	0
		1	1	0	1	0	1	0	1	0
		1	1	0	1	0	1	0	1	0
		0	0	0	0	0	0	0	0	0

Multiply

0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0

1	1	1	1	1	1	0	1
	1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	0	1
1	0	1	0	1	0	0	1	0

Add

Divide-and-Conquer Multiplication: Warmup

To multiply two n-digit integers:

- Multiply four ½n-digit integers.
- Add two ½n-digit integers, and shift to obtain result.

$$x = 2^{n/2} \cdot x_1 + x_0$$
$$y = 2^{n/2} \cdot y_1 + y_0$$
$$xy = \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right) = 2^n \cdot x_1 y_1 + 2^{n/2} \cdot \left(x_1 y_0 + x_0 y_1\right) + x_0 y_0$$

$$T(n) \leq 4T(n) \quad + \quad \Theta(n)$$

Recursive calls $\qquad$ add, shift

$$T(n) = \Theta(n^2)$$

$\uparrow$

assumes n is a power of 2

Karatsuba Multiplication

To multiply two n-digit integers:

- Add two $\tfrac{1}{2}n$ digit integers.
- Multiply three $\tfrac{1}{2}n$-digit integers.
- Add, subtract, and shift $\tfrac{1}{2}n$-digit integers to obtain result.

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0 \\
&= 2^n \cdot \underset{A}{x_1 y_1} + 2^{n/2} \cdot \big(\underset{B}{(x_1 + x_0)(y_1 + y_0)} - \underset{A}{x_1 y_1} - \underset{C}{x_0 y_0}\big) + \underset{C}{x_0 y_0}
\end{aligned}
$$

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

$$T(n) \le 3\, T\!\left(\frac{n}{2}\right) + \Theta(n)$$

$$T(n) = O\!\left(n^{\log 3}\right) = O(n^{1.585})$$

Matrix Multiplication

Matrix Multiplication

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

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

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

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 ½n-by-½n blocks.
- Conquer: multiply 8 ½n-by-½n recursively.
- Combine: add appropriate products using 4 matrix additions.

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

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

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$$

Recursive calls   *add, form submatrices*

$$T(n) = \Theta(n^3)$$

Matrix Multiplication: Key Idea

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

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

$$
\begin{aligned}
P_1 &= A_{11} \times (B_{12} - B_{22}) \\
P_2 &= (A_{11} + A_{12}) \times B_{22} \\
P_3 &= (A_{21} + A_{22}) \times B_{11} \\
P_4 &= A_{22} \times (B_{21} - B_{11}) \\
P_5 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\
P_6 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\
P_7 &= (A_{11} - A_{21}) \times (B_{11} + B_{12})
\end{aligned}
$$

$$
\begin{aligned}
C_{11} &= P_5 + P_4 - P_2 + P_6 \\
C_{12} &= P_1 + P_2 \\
C_{21} &= P_3 + P_4 \\
C_{22} &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

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

Fast Matrix Multiplication

Fast matrix multiplication. (Strassen, 1969)

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

Analysis.

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

$$T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2)$$

Recursive calls   *add, substract*

$$T(n) = \Theta\left(n^{log7}\right) = O(n^{2.81})$$

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 ~ 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]

$$\Theta(n^{\log_2 7}) = O(n^{2.81})$$

Q. Multiply two 2-by-2 matrices with only 6 scalar multiplications?

A. Impossible. [Hopcroft and Kerr, 1971]

$$\Theta(n^{\log_2 6}) = O(n^{2.59})$$

Q. Two 3-by-3 matrices with only 21 scalar multiplications?

A. Also impossible.

$$\Theta(n^{\log_3 21}) = O(n^{2.77})$$

Q. Two 70-by-70 matrices with only 143,640 scalar multiplications?

A. Yes!   [Pan, 1980]

$$\Theta(n^{\log_{70} 143640}) = O(n^{2.80})$$

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.]

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

Caveat. Theoretical improvements to Strassen are progressively less practical.