Today:

- Divide-and-conquer paradigm
- Strassen’s algorithm
- Polynomial multiplication

Merge Sort (review)

To sort n numbers:

1. if n=1, done,
2. recursively sort 2 lists of numbers with \([n/2]\) and \([n/2]\) elements
3. merge 2 sorted lists in \(\Theta(n)\) time

Structure of merge sort algorithm

- break problem into similar (smaller) subproblems
- recursively solve subproblems
- combine solutions to produce final answer

Divide-and-conquer paradigm

1. Divide problem into subproblems,
2. Conquer subproblems by solving recursively,
3. Combine subproblem solutions.
Merge sort as Divide-and-conquer algorithm

1. **Divide**: Divide $n$-array into two $n/2$-subarrays.
2. **Conquer**: Sort the two subarrays recursively.
3. **Combine**: Linear-time merge.

Recurrence for Merge sort

$$T(n) = \begin{cases} 
2 & T\left(\frac{n}{2}\right) \\
\Theta(n) & \text{work dividing \\ & \\ & combining} 
\end{cases}$$

$$T(n) = 2T(n/2) + \Theta(n)$$

Solve recurrence

review from last lecture

Master Theorem: $T(n) = aT(n/b) + f(n)$

1. $f(n) = O(n^{\log_b a - \epsilon}) \Rightarrow T(n) = O(n^\log_b a)$

2. $f(n) = \Theta(n^{\log_b a} \log^k n) \Rightarrow T(n) = \Theta(n^{\log_b a} n^{k+1})$

3. $f(n) = \Omega(n^{\log_b a + \epsilon})$ and $af(n/b) \leq cf(n) \Rightarrow T(n) = \Theta(f(n))$

Merge sort analysis

- write recurrence $T(n) = 2 \cdot T(n/2) + \Theta(n)$
- solve using MT

$a = 2$ and $b = 2 \Rightarrow n^{\log_2 2} = n \Rightarrow$

Case 2 of MT ($k = 0$) \Rightarrow

$$T(n) = \Theta(n \log n)$$
Binary search

Search a key in a sorted array

- Example Given array:

```
  3 5 7 8 9 12 15
```

Find key 9

Binary search — Divide-and-conquer algorithm

to find key in sorted array of \( n \) elements:

1. **Divide**: Check middle element.
2. **Conquer**: Search one subarray.
3. **Combine**: Trivial.

Recurrence for binary search

halve \( n \) on each iteration \( \Rightarrow \)

\# times halve \( n \) to get 1 is \( \lg n \)

\[
T(n) = \frac{\text{subproblem size}}{\text{# subproblems}} + \Theta(1)
\]

Solve recurrence using MT

\[
a = 1 \text{ and } b = 2 \Rightarrow n^{\log_2 1} = 0 \Rightarrow
\]

Case 2 of MT \((k = 0)\) \(\Rightarrow\)

\[
T(n) = \Theta(\lg n)
\]
Matrix multiplication

\[ A, B, C \in \mathbb{R}^{n \times n} \]

**Input:** \( A = (a_{ij}) \), \( B = (b_{ij}) \)

**Output:** \( C = (c_{ij}) = AB \)

\[ c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \] for \( i, j = 1, \ldots, n \)

\[
\begin{pmatrix}
  c_{ij}
  \\
  \vdots
\end{pmatrix} =
\begin{pmatrix}
  a_{ij}
  \\
  \vdots
\end{pmatrix} \cdot
\begin{pmatrix}
  b_{ij}
  \\
  \vdots
\end{pmatrix}
\]

\[ C = A \cdot B \]

Time to compute \( c_{ij} = \Theta(n) \)

Time to compute \( C = \Theta(n) \cdot n^2 = \Theta(n^3) \)

---

Divide-and-conquer algorithm

\( n \times n \) matrix = \( 2 \times 2 \) matrix of \( n/2 \times n/2 \) submatrices

\[
\begin{pmatrix}
  r & s \\
  t & u
\end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} e & g \\ f & h \end{pmatrix}
\]

\[ C = A \times B \]

\( A, B, C \in \mathbb{R}^{n \times n} \)

\( a, b, \ldots, f, h, \ldots, t, u \) all \( n/2 \times n/2 \) matrices

---

Reduction to subproblems of size \( n/2 \times n/2 \)

We can rewrite the equation

\[ C = A \times B \]

as

\[
\begin{pmatrix}
  r & s \\
  t & u
\end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} e & g \\ f & h \end{pmatrix}
\]

so that

\[
\begin{align*}
  r &= ae + bf \\
  s &= ag + bh \\
  t &= ce + df \\
  u &= cg + dh
\end{align*}
\]

One \( n \times n \) matrix multiplication is reduced to four equations, each with:

- 2 multiplications of \( n/2 \times n/2 \) matrices
- 1 addition of \( n/2 \times n/2 \) matrices

\( \Rightarrow \) Total: 8 Mults, 4 Adds
Analysis

(n is the dimension of matrix)

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

\[ T(n) = 8T(n/2) + \Theta(n^2) \]

\[ n^{\log_2 8} = n^3 \Rightarrow \text{Case 1 of MT} \]

\[ T(n) = \Theta(n^3) \quad \text{No better!} \]

Recursive algorithm

Idea: Multiply 2 \times 2 matrices with only 7 scalar multiplications (instead of 8)

Let

\[ P_1 = a \cdot (g - h) \]
\[ P_2 = (a + b) \cdot h \]
\[ P_3 = (c + d) \cdot e \]
\[ P_4 = d \cdot (f - e) \]
\[ P_5 = (a + d) \cdot (e + h) \]
\[ P_6 = (b - d) \cdot (f + h) \]
\[ P_7 = (a - c) \cdot (e + g) \]

then

\[ r = P_5 + P_4 - P_2 + P_6 \]
\[ s = P_1 + P_2 \]
\[ t = P_3 + P_4 \]
\[ u = P_5 + P_1 - P_3 - P_7 \]

Indeed,

\[ s = P_1 + P_2 \]
\[ = (ag - ah) + (ah + bh) \]
\[ = ag + bh \]

Check the rest!

Do not rely on commutativity of multiplication!
Same equations true for matrices instead of scalars.

Old method: 8 Mults, 4 Adds
New method: To compute \( P_1, \ldots, P_7, r, s, t, u \) need

- 7 multiplications
- 18 additions (and subtractions)

For \( n \times n \) matrices multiplication takes \( \Theta(n^3) \),
but addition takes \( \Theta(n^2) \).

14 more adds! — cost of eliminating 1 multiply
For scalars, not worth it.
For matrices, can be worth it.
Strassen’s algorithm

to multiply $n \times n$ matrices $C = AB$

1. **Divide:** Partition $A, B$ into $n/2 \times n/2$ matrices.
   Use $+$ and $-$ to form terms to be multiplied.

2. **Conquer:** Perform 7 multiplications recursively.

3. **Combine:** Form $C$ using $+$ and $-$.

**Analysis**

subproblem size

$$T(n) = \sum T\left( \frac{n}{2} \right)$$  

# multiplications

$$+ \Theta(n^2)$$  

work combining, i.e. adding

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

$$= \Theta(n^{\log_2 7}) \quad \text{(Case 1 of MT)}$$  

$$= O(n^{2.81})$$

$\Theta(n^{2.81})$ vs $\Theta(n^3)$, wins for $n \geq 50$ in practice

Best algorithm to date: $O(n^{2.376})$ (not practical!)

**Polynomial Multiplication**

$A, B \rightarrow n$-th order poly, $C \rightarrow 2n$-th order poly

$$A(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

$$B(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_n x^n$$

$$C(x) = A(x) \cdot B(x)$$

$$= c_0 + c_1 x + c_2 x^2 + \ldots + c_{2n} x^{2n}$$

$$c_i = \sum_{k=0}^{i} a_k b_{i-k} \quad \text{for } i = 0, \ldots, 2n$$

Convention: $a_j, b_j \equiv 0$ when $j > n$

Ordinary algorithm: $\Theta(n^2)$ running time.

**Example: $n = 2$**

$$A(x) = a_0 + a_1 x + a_2 x^2$$

$$B(x) = b_0 + b_1 x + b_2 x^2$$

$$C(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + c_4 x^4$$

$$c_0 = \sum_{k=0}^{0} a_k b_{0-k} = a_0 b_0$$

$$c_1 = \sum_{k=0}^{1} a_k b_{1-k} = a_0 b_1 + a_1 b_0$$

$$c_2 = \sum_{k=0}^{2} a_k b_{2-k} = a_0 b_2 + a_1 b_1 + a_2 b_0$$

$$c_3 = \sum_{k=0}^{3} a_k b_{3-k} = a_0 b_3 + a_1 b_2 + a_2 b_1 + a_3 b_0$$

$$c_4 = \sum_{k=0}^{4} a_k b_{4-k} = a_0 b_4 + a_1 b_3 + a_2 b_2 + a_3 b_1 + a_4 b_0$$

$$c_5 = \sum_{k=0}^{5} a_k b_{5-k} = a_0 b_5 + a_1 b_4 + a_2 b_3 + a_3 b_2 + a_4 b_1 + a_5 b_0$$
Write $C$ in $n/2$-th order polynomials

Factor $x^{n/2}$ out of latter terms of $A, B$:

$$A(x) = (a_0 + \ldots + a_{n/2-1}x^{n/2-1}) + x^{n/2}(a_{n/2} + a_{n/2+1}x + \ldots + a_nx^{n/2})$$
$$B(x) = R(x) + x^{n/2}S(x)$$
$$C(x) = P(x) \cdot R(x)$$
$$+x^{n/2}(P(x) \cdot S(x) + R(x) \cdot Q(x))$$
$$+x^nQ(x) \cdot S(x)$$

Divide-and-conquer algorithm

1. **Divide**: Partition $A, B$ into sum of a $n/2$-th order poly and $x^{n/2} \times (n/2$-th order poly).
2. **Conquer**: Perform 4 multiplications ($P \cdot R$,
   $P \cdot S$, $R \cdot Q$, and $Q \cdot S$) recursively.
3. **Combine**: Form $C$ using $+$ and $-$.

Analysis:

$$T(n) = 4T(n/2) + \Theta(n)$$
$$n^{\log_2 4} = n^2 \Rightarrow \text{Case 1 of MT}$$

$$T(n) = \Theta(n^2) \quad \text{No better!}$$

Clever divide-and-conquer

$$\frac{4 \text{ Mults!}}{(a + by)(c + dy) = \frac{ac + (ad + bc)y + bdy^2}{}}$$

Let

$$m_1 = (a + b) \cdot (c + d)$$
$$m_2 = a \cdot c$$
$$m_3 = b \cdot d$$

then don’t need 4th multiplication, since:

$$ad + bc = m_1 - m_2 - m_3$$

Let $a, b, c, d$ be polynomials, $y = x^{n/2}$

use divide-and-conquer to perform

only 3 multiplications recursively.

Analysis

$$T(n) = 3T(n/2) + \Theta(n)$$

$$n^{\log_2 3} = n^{1.59} \Rightarrow \text{Case 1 of MT}$$

$$T(n) = \Theta(n^{\log_3 3})$$