

Divide-and-conquer

(CLRS 4.2)

D&C is a powerful technique for solving problems:

Input: Problem P

To Solve P:

1. *Divide* P into smaller problems P_1, P_2
2. *Conquer* by solving the (smaller) subproblems recursively.
3. *Combine* solutions to P_1, P_2 into solution for P.

Matrix Multiplication

Let X and Y be two $n \times n$ matrices

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ x_{31} & x_{32} & \cdots & x_{3n} \\ \cdots & \cdots & \cdots & \cdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{pmatrix}$$

We want to compute $Z = X \cdot Y$, where $z_{ij} = \sum_{k=1}^n X_{ik} \cdot Y_{kj}$

Problem: Given two matrices of size n by n , come up with an algorithm to compute the product.

- The straightforward method uses $\Rightarrow n^2 \cdot n = \Theta(n^3)$ operations
- Can we do better? That is, is it possible to multiply two matrices faster than $\Theta(n^3)$?
- This was an open problem for a long time... until Strassen came up with an algorithm in 1969. The idea is to use divide-and-conquer.

Matrix multiplication with divide-and-conquer

- Let's imagine that n is a power of two. We can view each matrix as consisting of $2 \times 2 = 4$ $n/2$ -by- $n/2$ matrices.

$$X = \begin{pmatrix} A & B \\ C & D \end{pmatrix}, Y = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

- Then we see that their product $X \cdot Y$ can be written as:

$$\begin{Bmatrix} A & B \\ C & D \end{Bmatrix} \cdot \begin{Bmatrix} E & F \\ G & H \end{Bmatrix} = \begin{Bmatrix} (A \cdot E + B \cdot G) & (A \cdot F + B \cdot H) \\ (C \cdot E + D \cdot G) & (C \cdot F + D \cdot H) \end{Bmatrix}$$

- The above naturally leads to divide-and-conquer solution:
 - Divide X and Y into 8 sub-matrices A, B, C, D, E, F, G, H .
 - Compute 8 $n/2$ -by- $n/2$ matrix multiplications recursively.
 - Combine results (by doing 4 matrix additions) and copy the results into Z .
- ANALYSIS: Running time of algorithm is given by $T(n) = 8T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^3)$
- Cool idea, but not so cool result.....since we already discussed a (simpler/naive) $O(n^3)$ algorithm!
- Can we do better?

Strassen's divide-and-conquer

- Strassen's algorithm is based on the following observation:

The recurrence

$$T(n) = 8T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^3)$$

while the recurrence

$$T(n) = 7T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^{\lg 7})$$

- Strassen found a way to compute only 7 products of $n/2$ -by- $n/2$ matrices
- With same notation as before, we define the following 7 $n/2$ -by- $n/2$ matrices:

$$\begin{aligned} S_1 &= (B - D) \cdot (G + H) \\ S_2 &= (A + D) \cdot (E + H) \\ S_3 &= (A - C) \cdot (E + F) \\ S_4 &= (A + B) \cdot H \\ S_5 &= A \cdot (F - H) \\ S_6 &= D \cdot (G - E) \\ S_7 &= (C + D) \cdot E \end{aligned}$$

- Strassen observed that we can write the product Z as:

$$Z = \begin{Bmatrix} A & B \\ C & D \end{Bmatrix} \cdot \begin{Bmatrix} E & F \\ G & H \end{Bmatrix} = \begin{Bmatrix} (S_1 + S_2 - S_4 + S_6) & (S_4 + S_5) \\ (S_6 + S_7) & (S_2 + S_3 + S_5 - S_7) \end{Bmatrix}$$

- For e.g. let's test that $S_6 + S_7$ is really $C \cdot E + D \cdot G$

$$\begin{aligned}
S_6 + S_7 &= D \cdot (G - E) + (C + D) \cdot E \\
&= DG - DE + CE + DE \\
&= DG + CE
\end{aligned}$$

- This leads to a divide-and-conquer algorithm:
 - Divide X and Y into 8 sub-matrices A, B, C, D, E, F, G, H .
 - Compute $S_1, S_2, S_3, \dots, S_7$. This involves 10 matrix additions and 7 multiplications recursively.
 - Compute $S_1 + S_2 - S_4 + S_6, \dots$ and copy them in Z . This step involves only additions/subtractions of $n/2$ -by- $n/2$ matrices.
- ANALYSIS: $T(n) = 7T(n/2) + \Theta(n^2)$, with solution $O(n^{\lg 7})$.
- Lets solve the recurrence using the iteration method

$$\begin{aligned}
T(n) &= 7T(n/2) + n^2 \\
&= n^2 + 7(7T(\frac{n}{2^2}) + (\frac{n}{2})^2) \\
&= n^2 + (\frac{7}{2^2})n^2 + 7^2T(\frac{n}{2^2}) \\
&= n^2 + (\frac{7}{2^2})n^2 + 7^2(7T(\frac{n}{2^3}) + (\frac{n}{2^2})^2) \\
&= n^2 + (\frac{7}{2^2})n^2 + (\frac{7}{2^2})^2 \cdot n^2 + 7^3T(\frac{n}{2^3}) \\
&= n^2 + (\frac{7}{2^2})n^2 + (\frac{7}{2^2})^2n^2 + (\frac{7}{2^2})^3n^2 \dots + (\frac{7}{2^2})^{\log n - 1}n^2 + 7^{\log n} \\
&= \sum_{i=0}^{\log n - 1} (\frac{7}{2^2})^i n^2 + 7^{\log n} \\
&= n^2 \cdot \Theta((\frac{7}{2^2})^{\log n - 1}) + 7^{\log n} \\
&= n^2 \cdot \Theta(\frac{7^{\log n}}{(2^2)^{\log n}}) + 7^{\log n} \\
&= n^2 \cdot \Theta(\frac{7^{\log n}}{n^2}) + 7^{\log n} \\
&= \Theta(7^{\log n})
\end{aligned}$$

- Now we have the following:

$$\begin{aligned}
7^{\log n} &= 7^{\frac{\log_7 n}{\log_7 2}} \\
&= (7^{\log_7 n})^{(1/\log_7 2)}
\end{aligned}$$

$$\begin{aligned}
&= n^{(1/\log_7 2)} \\
&= n^{\frac{\log_2 7}{\log_2 2}} \\
&= n^{\log 7}
\end{aligned}$$

So the solution is $T(n) = \Theta(n^{\lg 7}) = \Theta(n^{2.81\dots})$

- Note:

- We are 'hiding' a much bigger constant in $\Theta()$ than before.
- Currently best known bound is $O(n^{2.376\dots})$ (Coppersmith and Winograd'78).
- Lower bound is (trivially) $\Omega(n^2)$.
- Big open problem!!
- Strassen's algorithm has been found to be efficient in practice once n is large enough. For small values of n the straightforward cubic algorithm is used instead. The crossover point where Strassen becomes more efficient depends from system to system.