Matrix multiplication: Strassen's algorithm

We've all learned the naive way to perform matrix multiplies in $O(n^3)$ time. In today's lecture, we review Strassen's sequential algorithm for matrix multiplication which requires $O(n^{\log_2 7}) = O(n^{2.81})$ operations;

Block Matrix Multiplication

The idea behind Strassen's algorithm is in the formulation of matrix multiplication as a recursive problem. We first cover a variant of the naive algorithm, formulated in terms of block matrices, and then parallelize it. Assume $A, B \in R^{n \times n}$ and $C = AB$, where n is a power of two.

We write A and B as block matrices,

$$A = \begin{pmatrix} A11 & A12 \\ A21 & A22 \end{pmatrix}$$

$$B = \begin{pmatrix} B11 & B12 \\ B21 & B22 \end{pmatrix}$$

$$C = \begin{pmatrix} C11 & C12 \\ C21 & C22 \end{pmatrix}$$

where block matrices A_{ij} are of size $n/2 \times n/2$ (same with respect to block entries of B and C). Trivially, we may apply the definition of block-matrix multiplication to write down a formula for the block-entries of C, i.e.

$$C11 = A11B11 + A12B21$$

$$C12 = A11B12 + A12B22$$

$$C21 = A21B11 + A22B21$$

$$C22 = A21B12 + A22B22$$

Parallelizing the Algorithm

Realize that A_{ij} and B_{kl} are smaller matrices, hence we have broken down our initial problem of multiplying two $n \times n$ matrices into a problem requiring 8 matrix multiplies between matrices of size $n/2 \times n/2$, as well as a total of 4 matrix additions.

we may come up with the following recurrence for work:

$$T(n) = 8T(n/2) + O(n^2)$$

By the Master Theorem, $T(n) = O(n^{\log_2 8}) = O(n^3)$.

Strassen's Algorithm

We now turn toward Strassen's algorithm, such that we will be able to reduce the number of sub-calls to matrix-multiplies to 7, using just a bit of algebra. In this way, we bring the work down to $O(n^{\log_2 7})$.

We write down C_{ij} 's in terms of block matrices M_k's. Each M_k may be calculated simply from products and sums of sub-blocks of A and B.

That is, we let

$$M1 = (A11 + A22)(B11 + B22)$$
$$M2 = (A21 + A22)B11$$
$$M3 = A11(B12 - B22)$$
$$M4 = A22(B21 - B11)$$
$$M5 = (A11 + A12)B22$$
$$M6 = (A21 - A11)(B11 + B12)$$
$$M7 = (A12 - A22)(B21 + B22)$$

It can be verified that

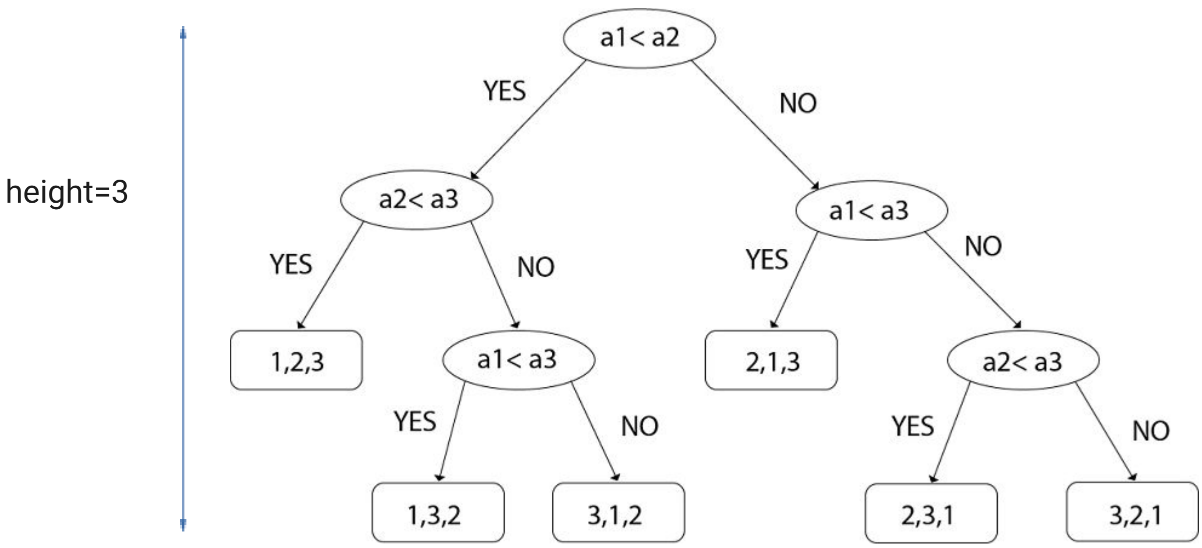
$$C11 = M1 + M4 - M5 + M7$$
$$C12 = M3 + M5$$
$$C21 = M2 + M4$$
$$C22 = M1 - M2 + M3 + M6$$

Realize that our algorithm requires quite a few summations, however, this number is a constant independent of the size of our matrix multiples. Hence, the work is given by a recurrence of the form

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

Lower Bound Theory for comparison sort

The preceding sections present three O(nlogn) sorting algorithms—quick sort, heap sort, and the two-way merge sort. But is O(nlogn) the best we can do?
In this section we answer the question by showing that any sorting algorithm that sorts using only binary comparisons must make Ω(nlogn) such comparisons.



Any sorting algorithm that uses only **binary comparisons** can be represented by a **binary decision tree**. Furthermore, it is the height of the binary decision tree that determines the worst-case running time of the algorithm.
Given an input sequence of **n items** to be sorted, every **binary decision tree** that correctly sorts the input sequence must have **at least n!** leaves--one for each permutation of the input. Therefore the **height of the binary decision tree** is **at least** $\lceil \log_2 n! \rceil$.

$$\begin{aligned} \lceil \log_2 n! \rceil &\geq \log_2 n! \\ &\geq \Sigma^{n}_{i=1} \log_2 i \\ &\geq \Sigma^{n/2}_{i=1} \log_2 i/2 \\ &\geq n/2 \log_2 n/2 \\ &= \Omega(n \log n) \end{aligned}$$