1. Analysis of matrix multiplication

We’ll start by defining an algorithm for multiplying matrices. Let’s assume that our matrices are square \((n \times n)\), and to make things even simpler, assume that \(n\) is a power of 2.

Let \(A\) and \(B\) be two \(n \times n\) matrices. We would like to compute the product \(AB\).

If \(n \geq 2\), we can view \(A\) and \(B\) as follows (each broken up into four \(\frac{n}{2} \times \frac{n}{2}\) matrices).

\[
A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}
\]

Then the product can be computed as follows:

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

If \(n = 1\), then if \(A = [a]\) and \(B = [b]\), \(AB = [ab]\).

This defines an algorithm for computing the product of two square matrices (when \(n\) is not a power of two, we just have to break the matrices up into 4 submatrices that are almost \(\frac{n}{2} \times \frac{n}{2}\)).

Let’s analyze the time complexity of this algorithm. Well, each submatrix is of size \(\frac{n}{2} \times \frac{n}{2}\). There are 8 matrix multiplications of matrices of size \(\frac{n}{2} \times \frac{n}{2}\). Furthermore, adding the matrices together will take \(dn^2\) steps for some constant \(d\) (because a matrix has \(n^2\) entries).

\[
T(n) = \begin{cases} c & n = 1 \\ 8T\left(\frac{n}{2}\right) + dn^2 & n > 1 \end{cases}
\]

Now we can find the closed form by repeated substitution:

\[
T(n) = 8T\left(\frac{n}{2}\right) + dn^2 \\
= 8\left(8T\left(\frac{n}{4}\right) + d\left(\frac{n}{2}\right)^2\right) + dn^2 = 8^2T\left(\frac{n}{4}\right) + (2 + 1)dn^2 \\
= 8^2\left(8T\left(\frac{n}{4}\right) + d\left(\frac{n}{4}\right)^2\right) + (2 + 1)dn^2 = 8^3T\left(\frac{n}{4}\right) + (4 + 2 + 1)dn^2 \\
= 8^3\left(8T\left(\frac{n}{4}\right) + d\left(\frac{n}{4}\right)^2\right) + (4 + 2 + 1)dn^2 = 8^4T\left(\frac{n}{4}\right) + (8 + 4 + 2 + 1)dn^2 \\
\vdots \\
= 8^iT\left(\frac{n}{2^i}\right) + \left(\sum_{j=0}^{i-1} 2^j\right)dn^2 \\
= 8^iT\left(\frac{n}{2^i}\right) + (2^i - 1)dn^2
\]

This expansion stops when \(i = \log_2 n\), since at this point, \(\frac{n}{2^i} = 1\), and thus we will be evaluating \(T(1)\) and can use the base case.

If we plug in \(i = \log_2 n\) to the above equation, we get:

\[
T(n) = 8^{\log_2 n}T\left(\frac{n}{2^{\log_2 n}}\right) + (2^{\log_2 n} - 1)dn^2 \\
= 2^{3\log_2 n}T\left(\frac{n}{n}\right) + (n - 1)dn^2 \\
= n^3T(1) + (n - 1)dn^2 \\
= cn^3 + dn^3 - dn^2 \\
= (c + d)n^3 - dn^2
\]

If we let \(k = c + d\), we see that \(T(n) \leq kn^3\).
An interesting algorithm was developed by Strassen. Instead of breaking it down into $8$ multiplications of matrices of size $\frac{n}{2} \times \frac{n}{2}$, he does just $7$. The recurrence for this algorithm looks like:

$$T(n) = \begin{cases} c & n = 1 \\ \frac{7}{7} T \left( \frac{n}{2} \right) + d n^2 & n > 1 \end{cases}$$

So what changes with the analysis? Well, we’ll end up with

$$T(n) = 7 \cdot T \left( \frac{n}{2} \right) + \left( \sum_{j=0}^{i-1} \frac{7}{4} \right) d n^2$$

Skipping the algebra of the second half... Look at the first half: $7^i = 7^{\log_2 n} = n^{\log_2 7} = n^{8.07}$.

This gives us a slightly better result than the previous algorithm! And the interesting thing is: We don’t know if we can do better! We of course can’t do better than $n^2$ (since there are $n^2$ entries in each matrix), but the optimal algorithm could be anywhere between $n^{2.807}$ and $n^2$!

For a more thorough discussion of Strassen’s algorithm, see Chapter 28.2 of Introduction to Algorithms by Cormen, Leiserson, Rivest, Stein. You have free access through the UToronto Library website: http://simplelink.library.utoronto.ca/url.cfm/1476

2. ONE FINAL RECURRANCE

One last recurrence to look at (which isn’t coming from an algorithm):

$$T(n) = \begin{cases} c & n = 1 \\ T \left( \frac{n}{2} \right) + d n & n > 1 \end{cases}$$

Repeated substitution yields:

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

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

$$= T \left( \frac{n}{2^3} \right) + d \left( \frac{n}{2^2} + n \right)$$

$$\vdots$$

$$= T \left( \frac{n}{2^i} \right) + d \left( \sum_{j=0}^{i-1} \frac{1}{2^j} \right) n$$

$$< T \left( \frac{n}{2} \right) + 2 d n$$

Once again, let $i = \log_2 n$. Then

$$T(n) < T(1) + 2 d = c + 2 d n$$

3. DIVIDE AND CONQUER IN GENERAL, VISITED

Don’t worry about this section too much, but it may give more meaning to the recurrence examples chosen. The three examples weren’t chosen at random. Recall the general form:

$$T(n) = \begin{cases} c & n = 1 \\ a T \left( \frac{n}{a} \right) + f(n) & n > 1 \end{cases}$$

Let’s just consider the case where $f$ is a polynomial, i.e., $f(n) = n^l$.

Divide and conquer recurrences fall into three categories:

1. $T(n) = \Theta(n^l)$. This happens when the combining time dominates, like it did in the final example above. The size of the problem is shrinking quickly and there aren’t too many copies at each level, so most (not exactly most, but you can think of it this way) of the time is spent in the first call of the recursive function. If you draw the recursion tree, the root is by far the biggest.

2. $T(n) = \Theta(n^{\log_a n})$. This is what we had with the matrix multiplication example (both the first algorithm and Strassen’s algorithm). If you draw the recursion tree, the leaves are where we spend most of our time. So what matters is how many times we have to call the base case. Since the height of the tree is $\log_a n$, the number of leaves is $a^{\log_a n} = n^{\log_a a}$.

3. $T(n) = \Theta(n^l \log n)$. This is our Goldilocks situation... right in the middle. Merge sort is an example of this one. Here, every level of the recursion tree takes the same amount of time ($n^l$). Since the height of the tree is $\log_n n$, we get $n^l \log_n n$. 

2
There is something called the *Master Theorem* which formally defines these cases. You can find it in Vassos’s course notes or in Chapter 4.3 of *Introduction to Algorithms* by CLRS (http://simplelink.library.utoronto.ca/url.cfm/1476).

Please DON’T memorize this! It’s nice to be aware of it’s existence, but you can always just solve by repeated substitution (the Master Theorem is proven by repeated substitution).