Flop

flop is an acronym for floating point operation. Counting the number of flops an algorithm requires to solve a problem allows us to compare – at least roughly – the relative speed of methods. (The term *flops* is also used by computer marketers to mean "floating point operations per second", and is a measure of the speed of the computer).

In older papers and textbooks flop stood for an operation like $c_{ij} = s + a_{ik} * b_{kj}$, which included a floating point multiply, a floating point add, and some indexing work. Today (since the mid 1990's) flop means one floating point operation $(+,\ -,\ *,\ /,\ \surd,\ >$, etc.). (The older version of flop is thus about 2 of the newer flops). One could argue that $+$ is faster than, e.g. $\surd$, and thus shouldn't be counted the same. This is true, and if an algorithm had a relatively large amount of $\surd$'s, then we would count them separately. In practice, counting flops gives only a rough way to compare algorithms, and since a $>$ and a $\surd$ each require bringing floats into registers and storing a result, this generic perspective will serve us well.

An algorithm for a problem of size n has *polynomial time* complexity if it requires at most $p(n)$ operations to complete its task, where p is some fixed polynomial. The algorithms we will study here usually have polynomial time complexity, and in the matrix contexts, p is often cubic. If an algorithm has complexity $p(n) = an^3 + bn^2 + c$, we will usually talk about it as $p(n) = an^3 + \mathrm{O}(n^2)$, since for large n the leading term is much larger than the rest. Think of $\mathrm{O}(\textit{something})$ as a quantity that is small enough to be ignored but, with a nod to "=", not forgotten. More precisely,

$$f(n) = \mathrm{O}(g(n)) \text{ if } \exists k, N > 0 \text{ such that } n > N \Rightarrow |f(n)| \leq k|g(n)|.$$

It will be typical for us to take $g(n) = n^s,\ \text{ for } s = 1,\ 2,\ \text{ or } 3.$

Aside: we have also seen this notation with small quantities (think Taylor's thm, or rounding error analysis), in that case the perspective is the same, but we have

$$f(h) = \mathrm{O}(g(h)) \text{ if } \exists k, \delta > 0 \text{ such that } h < \delta \Rightarrow |f(h)| \leq k|g(h)|,$$

and here also a typical g is $g(h) = h^s$.

As an example consider, the complexity of matrix multiplication. If $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$, then $C = AB$ is an $m \times p$ matrix, and each element can be computed as a dot product $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$. Each c_{ij} requires n multiplies and $n - 1$ adds, so the complexity for the product is $(2n - 1)mp = 2mnp + \mathrm{O}(mp) = \mathrm{O}(mnp)$ flops. Notice that the flop count is associated with the algorithm, not the problem. (The complexity of a problem is the complexity of the fastest algorithm for that problem). For example, consider the complexity of computing ABC for rectangular matrices by studying the two "algorithms"$(AB)C$ and $A(BC)$.

You might find it interesting to imagine (or construct) a table comparing $n, n\log_2(n), n^2, n^3, 2^n, n!$ for $n = 8, 16, 64, 512$.