# CATAM Lecture Notes
# Complexity

We often have a choice of algorithms to perform some calculation, and it is useful to be able to compare how long the different algorithms might take. We use the *complexity* of the algorithms — expressed in terms of one or more parameters such as $N$, the number of steps in an ODE integration, or the size of a matrix — to make the comparison. The complexity is measured in terms of how many operations (or "steps") the computer needs to perform in order to complete the algorithm.

To do this we need to choose a definition for an *operation*. In making a calculation involving additions and multiplications, for example, we usually count the multiplications as "operations" and ignore the additions, because they take so little time in comparison. However, in an algorithm which performs thousands of additions and only a few multiplications, this would be a silly definition. Therefore the definition of an operation varies from case to case, and you should always choose your definition carefully and state it.

**Example:** Suppose we calculate $\sum_{r=1}^{n} r^3$. This involves $2n$ multiplications (two for each $r^3$), and $n-1$ additions, which we ignore. The complexity is said to be $O(n)$; i.e., the time taken will be proportional to $n$ for large $n$.

If instead we use the formula $\frac{1}{4}n^2(n+1)^2$ we only need 4 multiplications and the complexity is $O(1)$, i.e., not dependent on $n$.

The actual *time* which a computer takes to perform a given algorithm will vary from machine to machine — the constant of proportionality in the example above will vary — so the idea of complexity gives us a machine-independent way of comparing algorithms. It is quite possible for an algorithm of complexity $O(n)$ to run more quickly on Computer A than an algorithm of complexity $O(1)$ does on Computer B, for a specific given value of $n$, if Computer A is much quicker than the other. The complexity only tells us which algorithm is better on a given machine.

Notice also that an $O(n^2)$ algorithm may be quicker for some values of $n$ than an $O(n)$ one. Suppose that the former requires $\frac{1}{2}n^2$ operations and the latter $10n$; then the first algorithm is quicker for $n < 20$. Complexity only tells us which is quicker for large values of $n$ (i.e., as $n \to \infty$).

**Example:** Multiplying an $m \times n$ matrix by an $n \times p$ matrix. For each of the $mp$ entries in the result we need to perform $n$ multiplications (and several additions, which we ignore); so the complexity is $O(mnp)$. In particular, to multiply two $n \times n$ matrices, the complexity is $O(n^3)$. In fact, there is an algorithm which takes only $O(n^{\log_2 7})$ multiplications!

**Example:** Sorting a list of numbers (or names). Here the basic operation is (often) a *comparison*. The most efficient algorithm can be proved to require $O(n \log n)$ comparisons; simple alternatives like "Bubblesort" require $O(n^2)$.

Most algorithms have some kind of "overheads". For instance, suppose we wish to calculate

$$\left(\lfloor \sqrt[3]{n} \rfloor\right)!$$

where $\lfloor \ldots \rfloor$ denotes the integer part. We would need to calculate $m = \lfloor \sqrt[3]{n} \rfloor$ once; this takes $O(1)$ steps, i.e., it is not dependent on $n$. (Depending on the way the computer

calculates the root, this may not be strictly true, but the time taken will certainly be only weakly dependent on $n$.) Then we must calculate $m!$, which involves $m-2$ multiplications. We will also need a loop counter, but the costs of this will be much less than the multiplications. Overall the algorithm takes $O(n^{1/3})$ steps; for large $n$, which is all that interests us in a complexity calculation, the $O(1)$ steps to calculate $m$ are negligible, and the difference between $m-2$ and $m$ can be ignored (as can the $\lfloor \ldots \rfloor$ operation and the loop counting).

Similarly, an algorithm taking $4 + 5n + \frac{1}{3}n^3$ operations would be referred to as $O(n^3)$.

Finally, note the notations $\Omega$ and $\Theta$ which may sometimes be used instead of $O$. A function $f(n)$ is said to be $\Omega\big(g(n)\big)$ iff $g(n) = O\big(f(n)\big)$; i.e., $\Omega$ is (in some sense) the opposite of $O$. Furthermore, $f(n)$ is said to be $\Theta\big(g(n)\big)$ iff both $f(n) = O\big(g(n)\big)$ and $g(n) = O\big(f(n)\big)$. Hence:

$$\begin{gathered} 3n = O(n); \qquad 3n = O(n^2); \qquad 3n \neq O(\sqrt{n}); \\ n^2 = \Omega(n); \qquad \tfrac{1}{2}n^2 + n = \Omega(n^2); \qquad n \neq \Omega(n^2); \\ 2n^3 + 2 = \Theta(n^3); \qquad 3n^2 \neq \Theta(n^3). \end{gathered}$$

Informally, $O$ means "same order of magnitude or less for large $n$"; $\Omega$ means "same order of magnitude or greater"; and $\Theta$ means "same order of magnitude".