(c) If $f(n)$ denotes the number k of binary digits in n, then it follows from the above formulas for k that $f(n) = O(\log n)$. Also notice that the same relation holds if $f(n)$ denotes the number of base-b digits, where b is any fixed base. On the other hand, suppose that the base b is not kept fixed but is allowed to increase, and we let $f(n, b)$ denote the number of base-b digits. Then we would want to use the relation $f(n, b) = O(\frac{\log n}{\log b})$.

(d) We have: $\text{Time}(n \cdot m) = O(\log n \cdot \log m)$, where the left hand side means the number of bit operations required to multiply n by m.

(e) In Exercise 6, we can write: $\text{Time}(n!) = O\big((n \log n)^2\big)$.

(f) In Exercise 7, we have:

$$\text{Time}\Big(\sum a_i x^i \cdot \sum b_j x^j\Big) = O\Big(n_1 n_2 \big((\log m)^2 + \log(\min(n_1, n_2))\big)\Big).$$

In our use, the functions $f(n)$ or $f(n_1, n_2, \ldots, n_r)$ will often stand for the amount of time it takes to perform an arithmetic task with the integer n or with the set of integers $n_1, n_2, \ldots, n_r$ as input. We will want to obtain fairly simple-looking functions $g(n)$ as our bounds. When we do this, however, we do not want to obtain functions $g(n)$ which are much larger than necessary, since that would give an exaggerated impression of how long the task will take (although, from a strictly mathematical point of view, it is not incorrect to replace $g(n)$ by any larger function in the relation $f = O(g)$).

Roughly speaking, the relation $f(n) = O(n^d)$ tells us that the function f increases approximately like the d-th power of the variable. For example, if $d = 3$, then it tells us that doubling n has the effect of increasing f by about a factor of 8. The relation $f(n) = O(\log^d n)$ (we write $\log^d n$ to mean $(\log n)^d$) tells us that the function increases approximately like the d-th power of the number of binary digits in n. That is because, up to a constant multiple, the number of bits is approximately $\log n$ (namely, it is within 1 of being $\log n/\log 2 = 1.4427 \log n$). Thus, for example, if $f(n) = O(\log^3 n)$, then doubling the number of bits in n (which is, of course, a much more drastic increase in the size of n than merely doubling n) has the effect of increasing f by about a factor of 8.

Note that to write $f(n) = O(1)$ means that the function f is bounded by some constant.

Remark. We have seen that, if we want to multiply two numbers of about the same size, we can use the estimate $\text{Time}(k\text{-bit}\cdot k\text{-bit}) = O(k^2)$. It should be noted that much work has been done on increasing the speed of multiplying two k-bit integers when k is large. Using clever techniques of multiplication that are much more complicated than the grade-school method we have been using, mathematicians have been able to find a procedure for multiplying two k-bit integers that requires only $O(k \log k \log\log k)$ bit operations. This is better than $O(k^2)$, and even better than $O(k^{1+\epsilon})$ for any $\epsilon > 0$, no matter how small. However, in what follows we shall always