plications and n_2 additions. The numbers being multiplied are bounded by m, and the numbers being added are each at most m^2; but since we have to add the partial sum of up to n_2 such numbers we should take n_2m^2 as our bound on the size of the numbers being added. Thus, in computing the coefficient of x^ν the number of bit operations required is at most

$$(n_2+1)(log_2m+1)^2+n_2(log_2(n_2m^2)+1).$$

Since there are n_1+n_2+1 values of ν, our time estimate for the polynomial multiplication is

$$(n_1+n_2+1)\big((n_2+1)(log_2m+1)^2+n_2(log_2(n_2m^2)+1)\big).$$

A slightly less rigorous bound is obtained by dropping the 1's, thereby obtaining an expression having a more compact appearance:

$$\frac{n_2(n_1+n_2)}{log\,2}\left(\frac{(log\,m)^2}{log\,2}+(log\,n_2+2\log\,m)\right).$$

Remark. If we set $n=n_1\geq n_2$ and make the assumption that $m\geq 16$ and $m\geq\sqrt{n_2}$ (which usually holds in practice), then the latter expression can be replaced by the much simpler $4n^2(log_2m)^2$. This example shows that there is generally no single "right answer" to the question of finding a bound on the time to execute a given task. One wants a function of the bounds on the imput data (in this problem, n_1, n_2 and m) which is fairly simple and at the same time gives an upper bound which for most input data is more-or-less the same order of magnitude as the number of bit operations that turns out to be required in practice. Thus, for example, in Example 7 we would not want to replace our bound by, say, $4n^2m$, because for large m this would give a time estimate many orders of magnitude too large.

So far we have worked only with addition and multiplication of a k-bit and an ℓ-bit integer. The other two arithmetic operations — subtraction and division — have the same time estimates as addition and multiplication, respectively: Time(subtract k-bit from ℓ-bit)$\leq \max(k,\ell)$; Time(divide k-bit by ℓ-bit)$\leq kl$. More precisely, to treat subtraction we must extend our definition of a bit operation to include the operation of subtracting a 0- or 1-bit from another 0- or 1-bit (with possibly a "borrow" of 1 from the previous column). See Exercise 8.

To analyze division in binary, let us orient ourselves by looking at an illustration, such as the one in Example 3. Suppose $k\geq\ell$ (if $k<\ell$, then the division is trivial, i.e., the quotient is zero and the entire dividend is the remainder). Finding the quotient and remainder requires at most $k-\ell+1$ subtractions. Each subtraction requires ℓ or $\ell+1$ bit operations; but in the latter case we know that the left-most column of the difference will always be a 0-bit, so we can omit that bit operation (thinking of it as "bookkeeping" rather than calculating). We similarly ignore other administrative details, such as the time required to compare binary integers (i.e., take just enough