In section 14.2 it is shown how to multiply two N-bit numbers using $O(N^{\log_2(3)}) \approx O(N^{1.585})$ bitwise operations, instead of the N^2 that standard long multiplication takes. The explanation is somewhat overly concise, so here I will show all the steps.

The problem is how to multiply A times B to get C. A and B are N-bit numbers, and we shall assume that N is a power of two.

The first step is to divide A into two halves, which we call the left (most significant) half A_L, and the right (least significant) half A_R. For example, if $N = 8$ and $A = 137$, then in bits we have $A = 10001001$, so that $A_L = 1000$ and $A_R = 1001$. Note that in this example $A = A_L \text{shl} 4 \ + \ A_R$, where shl stands for the left linear shift operator, which is the same as multiplying by $2^4 = 16$, $A = 16A_L \ + \ A_R$. More generally

$$A \quad = \quad A_L \text{ shl } \frac{N}{2} \ + \ A_R \quad = \quad A_L 2^{\frac{N}{2}} \ + \ A_R$$

and similarly dividing B into two halves, leads us to

$$B \quad = \quad B_L \text{ shl } \frac{N}{2} \ + \ B_R \quad = \quad B_L 2^{\frac{N}{2}} \ + \ B_R$$

which are the first two lines of equation 14.1.

Multiplying A times B gives

$$C = (A_L 2^{\frac{N}{2}} + A_R)\,(B_L 2^{\frac{N}{2}} + B_R) = A_L B_L 2^N + (A_L B_R + A_R B_L) 2^{\frac{N}{2}} + A_R B_R$$

which consists of 4 products of $\frac{N}{2}$-bit numbers, and so takes $4\frac{N}{2}^2 = N^2$ bitwise multiplications, and so we have not gained anything. However, it is easy to convince oneself that this is the same as the expression in equation 14.1

$$C = A_L B_L (2^N + 2^{\frac{N}{2}}) + (A_L - A_R)(B_R - B_L) 2^{\frac{N}{2}} + A_R B_R (2^{\frac{N}{2}} + 1)$$

(it may be hard to figure out what to add and subtract to arrive at this formula, but it *is* easy to check).

As stated in the book, this expression for C involves only three multiplications of $\frac{N}{2}$-length numbers (well, actually $(A_L - A_R)$ and $(B_R - B_L)$ can be $\frac{N}{2} + 1$ bits in length, but let's neglect that). So, since these multiplications are certainly be carried out using standard long multiplication, each takes $(\frac{N}{2})^2$ bitwise multiplication operations, so that the three together require $3(\frac{N}{2})^2 = \frac{3}{4}N^2$ bitwise multiplies. This is 25% less than standard long multiplication.

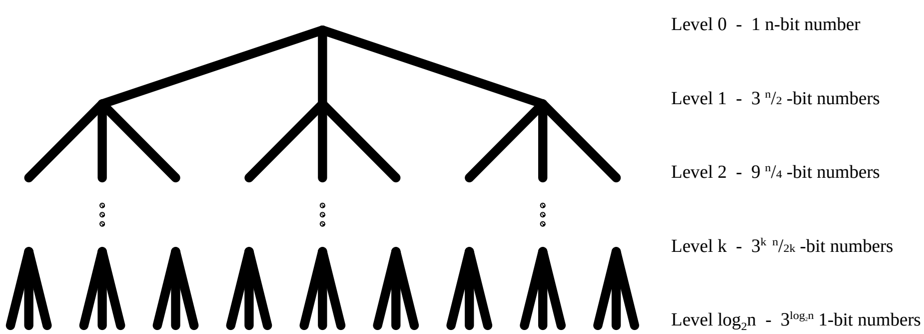
Note that we are not counting the multiplications by 2^N or $2^{\frac{N}{2}}$ as these are just bit shifts, not true multiplications. Similarly, we are ignoring the addition operations.

We save 25% *if* we carry out the three multiplications using standard long multiplication. However, there is no reason to multiply A_L by B_L in this way, after we have learned the trick. Instead, we divide A_L into A_{LL} and A_{LR} such that $A_L = A_{LL}2^{\frac{N}{4}} + A_{LR}$ and similarly for B_L. We do the same for the other two products as well. This reduces the complexity by a further 25%.

Now, since we assumed that N is a power of two, we can continue dividing the parts of A ad B until we get to single bits. Since we know how to multiply single bits, we are done.

How many times do we have to divide the numbers in half ? Well, we started with N bits, after the first split we have numbers with $\frac{N}{2}$ bits, after the second we are left with numbers with $\frac{N}{4}$ bits, etc. It is easy to see that we need to do this $\log_2 N$ times. For example, for $N = 8$, $\log_2 8 = 3$. The first division leads to 4-bit numbers, the second to 2-bit numbers, and indeed the third to 1-bit numbers.

How many bitwise multiplications are left to be performed at the end? Well, since at the k^{th} level we have 3^k multiplications, at the final $(\log_2 N)^{\text{th}}$ level we have $3^{\log_2 N}$ of them. All of this is made clearer in the figure.



We found that we need $3^{\log_2 N}$ operations. It is surprising that $3^{\log_2 N} = N^{\log_2 3}$ (didn't you learn that in high-school?). Let's see why. N can always be written $2^{\log_2 N}$, so

$$\begin{aligned}
N^{\log_2 3} &= 2^{(\log_2 N)^{(\log_2 3)}} \\
&= 2^{(\log_2 N)(\log_2 3)} \\
&= 2^{(\log_2 3)(\log_2 N)} \\
&= 2^{(\log_2 3)^{(\log_2 N)}} \\
&= 3^{\log_2 N}
\end{aligned}$$

as promised.

So we have proven that by successively dividing the factors in two, we can reduce the complexity from $O(N^2)$ to $O(N^{\log_2(3)}) \approx O(N^{1.585})$.

However, the complexity can be further reduced by using the FFT. To see why, let's rename the numbers we wish to multiply a and b, to emphasize that they are in a time domain representation. For example, if we wish to multiply 137 times 85, then a is the time domain sequence 10001001, and b is 01010101. We saw that to multiply these in the time domain requires a convolution, which means we need to shift a versus b and at each shift multiply the corresponding elements. This is what leads to the N^2 complexity. However, we can convert a to A (the frequency domain representation) in $N \log_2 N$ operations, and similarly b to B. The multiplication of A and B is not a convolution, but simply a bit by bit multiplication (with no carries!). This takes only N operations to perform. Finally we have to convert the resultant C in the frequency domain, back to c in the time domain, at the cost of another $N \log_2 N$ operations. Altogether we have $3O(N \log_2 N) + O(N)$ which is $O(N \log N)$.