I'm not supplying answers to the parts that most of you got right.

Question 1

The unambitious way to tackle this question is to work out a few exampls, and see if a pattern emerges. It might. The chief drawback is that the number of examples you have to calculate to get any sort of a picture is large enough for there to be a reasonable possibility of you making a mistake *en route* and that could altogether spoil the effect. But in general the strategy of calculating a few special cases to get a feel is not a bad idea.

The ambitious way is to try to work out the answer by thinking it through. $|w_1|$ and $w_2|$ are the lengths of the words w_1 and w_2. How long is a word obtained by interleaving w_1 and w_2? Clearly such a word has $|w_1| + |w_2|$ places where characters might be, and $|w_1|$ of those are occupied by characters from w_1. Clearly there are $\binom{|w_1|+|w_2|}{|w_1|}$ ways of picking those $|w_1|$ places to be occupied by characters from w_1. Once you have decided which places are to be occupied by characters from w_1 there is only one way of filling in those characters from w_1—since they have to be inserted in order—so there are $\binom{|w_1|+|w_2|}{|w_1|}$ ways of interleaving w_1 with w_2.

Of course we could also have thought of this as interleaving w_2 with w_1 and then we would have obtained the answer $\binom{|w_2|+|w_1|}{|w_2|}$ which is of course the same (whew!) namely

$$\frac{(|w_1| + |w_2|)!}{|w_1|! \times |w_2|!}$$

Question 2

Starting from a machine $\mathfrak{M}_1$ that recognises L_1 and $\mathfrak{M}_2$ that recognises L_2, how do we build a machine that recognises an interleaving of words from L_1 and L_2? A machine that does this must somehow be keeping two strings in mind, a string that is perhaps going to become something in L_1 and a string that is perhaps going to become something in L_2. Let's start, as the hint suggests, by considering the easy case where L_1 and L_2 come from disjoint alphabets. (L_1 might contain strings of letters, and L_2 contain strings of numerals.) This makes it easy beco's—when a new character comes along for the machine to process—there need be no doubt in anybody's mind which of the two strings the character belongs to. Thus it is clear that—if the alphabets of L_1 and L_2 are disjoint—the machine you want for recognising interleavings consists of the two machines $\mathfrak{M}_1$ and $\mathfrak{M}_2$ next to each other, and when a character comes along you feed it to whichever machine is appropriate.

How do you formally describe a machine that consists of $\mathfrak{M}_1$ and $\mathfrak{M}_2$ sitting together side by side? You have to say what its states and transitions are. To specify its states, you have to specify what states $\mathfrak{M} - 1$ and $\mathfrak{M}_2$ are in , which is to say that its states are ordered pairs of states, the first component being a state of $\mathfrak{M}_1$ and the second component being a state of $\mathfrak{M}_2$. What is the

transition relation? Well, suppose our new machine is in state $\langle s_1, s_2 \rangle$, and receives a character c. If c is from the alphabet for L_1 then we move to a state $\langle t_1, s_2 \rangle$—where t_1 is the state to which $\mathfrak{M}_1$ moves if it is in state s_1 and receives character c. On the other hand if c is from the alphabet for L_2 then we move to a state $\langle s_1, t_2 \rangle$—where t_2 is the state to which $\mathfrak{M}_2$ moves if it is in state s_2 and receives character c. The start state of the new machine is the ordered pair of the two start states, and a state is accepting if both its components are accepting.,

This is a description of a deterministic machine, but the reason why it is deterministic is that the alphabets of L_1 and L_2 are disjoint, so that whenever a character arrives we can tell which alphabet it belongs to and we know which coordinate of the ordered pair to alter. If the alphabets for L_1 and L_2 overlap—or are the same—then we don't know whether to think of the incoming character as being part of a string in L_1 or part of a string in L_2.

This means that, if the character c that the machine receives when in state $\langle s_1, s_2 \rangle$ belongs to both alphabets then we must put in *two* transitions, one to a state $\langle t_1, s_2 \rangle$—where t_1 is the state to which $\mathfrak{M}_1$ moves if it is in state s_1 and receives character c **and** one to a state $\langle s_1, t_2 \rangle$—where t_2 is the state to which $\mathfrak{M}_2$ moves if it is in state s_2 and receives character c. This machine is nondeterministic. However its degree of nondeterminism (see the hint) is 2.

Question 3

Once you are happy with that it will be fairly clear what to do with interleavings of more than two languages. If you interleave n languages $L_1 \dots L_n$, recognised respectively by machines $\mathfrak{M}_1 \dots \mathfrak{M}_n$, then the machine that recognises interleavings of the languages $L_1 \dots L_n$ has states that are ordered n-tuples of states of machines $\mathfrak{M}_1 \dots \mathfrak{M}_n$. And its degree of nondeterminism will be n.