Properties of recursive algorithms

Here is the basic idea behind recursive algorithms:

> To solve a problem, solve a subproblem that is a smaller instance of the same problem, and then use the solution to that smaller instance to solve the original problem.

When computing **n!**, we solved the problem of computing **n!** (the original problem) by solving the subproblem of computing the factorial of a smaller number, that is, computing **(n-1)!** (the smaller instance of the same problem), and then using the solution to the subproblem to compute the value of **n!**.
In order for a recursive algorithm to work, the smaller subproblems must eventually arrive at the base case. When computing **n!**, the subproblems get smaller and smaller until we compute **0!**. You must make sure that eventually, you hit the base case.
For example, what if we tried to compute the factorial of a negative number using our recursive method? To compute **(-1)!**, you would first try to compute **(−2)!**, so that you could multiply the result by **−1**. But to compute **(−2)!**, you would first try to compute **(−3)!**, so that you could multiply the result by **−2**. And then you would try to compute **(−3)!**, and so on. Sure, the numbers are getting smaller, but they're also getting farther and farther away from the base case of computing **0!**. You would never get an answer.

Even if you can guarantee that the value of **n** is not negative, you can still get into trouble if you don't make the subproblems progressively smaller. Here's an example. Let's take the formula **n!** = **n**·**(n−1)!** and divide both sides by **n**, giving **n!**/**n** = **(n−1)!**. Let's make a new variable, **m**, and set it equal to **n**+**1**. Since our formula applies to any positive number, let's substitute **m** for **n**, giving **m!**/**m** = **(m−1)!**. Since **m** = **n** + **1**, we now have **(n+1)!** / **(n+1)** = **(n+1-1)!**. Switching sides and noting that **n+1-1** = **n** gives us **n!** = **(n+1)!** / **(n+1)**. This formula leads us to believe that you can compute **n!** by first computing

(n+1)! and then dividing the result by **n+1**. But to compute **(n+1)!**, you would

have to compute **(n+2)!**, then **(n+3)!**, and so on. You would never get to the base case of 0. Why not? Because each recursive subproblem asks you to compute the value of a larger number, not a smaller number. If **n** is positive, you would never hit the base case of 0.

We can distill the idea of recursion into two simple rules:

1. Each recursive call should be on a smaller instance of the same problem, that is, a smaller subproblem.
2. The recursive calls must eventually reach a base case, which is solved without further recursion.

Let's go back to the Russian dolls. Although they don't figure into any algorithms, you can see that each doll encloses all the smaller dolls (analogous to the recursive case), until the smallest doll that does not enclose any others (like the base case).