Step 4: Test Your Algorithm

After Step 3, we have an algorithm that we *think* is right. However, it is entirely possible that we have messed up along the way. The primary purpose of Step 4 is to ensure our steps are actually right before we proceed. To accomplish this, we test our algorithm with *different* values of the parameters than the ones we used to design our algorithm. We execute our algorithm by hand and compare the answer it obtains to the right answer. If they differ, then we know our algorithm is wrong. The more test cases (values of parameters) we use, the more confident we can become that our algorithm is correct. Unfortunately, it is impossible to ensure that our algorithm is correct by testing. The only way to be completely sure that your algorithm is correct is to formally prove its correctness (using a mathematical proof), which is beyond the scope of this specialization.

One common type of mistake is mis-generalizing in Step 3. As we just discussed, one might think that the steps repeated x times because $x = 3$ and the steps repeated 3 times. If we had written that down in Step 3, our algorithm would only work when $x = y - 1$; otherwise we would count the wrong number of times and get the wrong answer. If that were the case, we would *hopefully* detect the problem by testing our algorithm by hand in Step 4. When we detect such a problem, we must go back and re-examine the generalizations we made in Step 3. Often, this is best accomplished by returning to Steps 1 and 2 for whatever test case exposed the problem. Re-doing Steps 1 and 2 will give you a concrete set of steps to generalize differently. You can then find where the generalization you came up with before is wrong, and revise it accordingly.

Another common type of mistake is that there are cases we did not consider in designing our algorithm. In fact, in our x^y example, we did not consider what happens when $y = 0$, and our algorithm handles this case incorrectly. If you execute the algorithm by hand with $x = 2$, $y = 0$, you *should* get 2^0=1; however, you will get an answer of 2. Specifically, you will start with $n = x = 2$. We would then try to count up from 1 to $0 - 1 = -1$, of which there are no numbers, so we would be done counting right away. We would then give back n (which is 2) as our answer.

To fix our algorithm, we would go back and revisit Steps 1 and 2 for the case that failed ($x = 2$, $y = 0$). This case is a bit tricky since we just know that the answer is 1 without doing any work ($x^0=1$ for any x). The fact that the answer requires no work makes Step 2 a little different—we just give an answer of 1. While this simplicity may seem nice, it actually makes it a little more difficult to incorporate it into our generalized steps. We might be tempted to write generalized steps like these:

```
If y is 0 then
  1 is your answer
Otherwise:
  Start with n = x
  Count up from 1 to y-1 (inclusive), for each number you count,
    n = Multiply x by n
  n is your answer.
```

These steps check explicitly for the case that gave us a problem ($y = 0$), give the right answer for that case, then perform the more general algorithm. For some problems, there may be corner cases which require this sort of special attention. However, for this problem, we can do better. Note that if you were unable to see the better solution and were to take the above approach, it is not wrong *per se*, but it is not the best solution.

Instead, a better approach would be to realize that if we count no times, we need an answer of 1, so we should start n at 1 instead of at x. In doing so, we need to count 1 more time (to y instead of to $y - 1$)—to multiply by x one more time:

```
Start with n = 1
Count up from 1 to y (inclusive), for each number you count,
  n = Multiply x by n
n is your answer.
```

Whenever we detect problems with our algorithm in Step 4, we typically want to return to Steps 1 and 2 to get more information to generalize from. Sometimes, we may see the problem right away (*e.g.*, if we made a trivial arithmetic mistake, or if executing the problematic test case by hand gives us insight into the correct generalization). If we see how to fix the problem, it is fine to fix it right away without redoing Steps 1 and 2, but if you are stuck, you should redo those steps until you find a solution. Whatever approach you take to fixing your algorithm, you should re-test it with all the test cases you have already used, as well as some new ones.

Determining good test cases is an important skill that improves with practice. For testing in Step 4, you will want to test with cases which at least produce a few different answers (*e.g.*, if your algorithm has a "yes" or "no" answer, you should test with parameters which produce both "yes" and "no"). You should also test any *corner cases*—cases where the behavior may be different from the more general cases. Whenever you have conditional decisions (including limits on where to count), you should test potential corner cases right around the boundaries of these conditions. For example, if your algorithm makes a decision based on whether or not $x < 3$, you might want to test with $x = 2$, $x = 3$, and $x = 4$. You can limit your "pencil and paper" testing somewhat, since you will do more testing on the actual code once you have written it.