Programs not working for larger parameters

Paul Stankovski

It is quite common for a program to appear fully functional for smaller problem instances, and it is also fully normal for it to blow up in your face when you run that larger problem instance that you need to solve. Why is that?

1 The Problem

The underlying problem is that your implementation is not perfect. You have introduced one or more bugs into your code.

The key word above is appear. Your program only *appeared* to be fully functional when you ran it for the smaller problem instances.

You might be tempted to say

I know that the code is fine, it is working for all smaller parameters.

You cannot claim that your code is correct only because it works for the smaller problem instances. Instance size matters. Things happen in a program when you increase the size of the problem instance. Unfortunately, working for smaller instances does not imply working for larger instances.

For example, data types can be an issue. If you used a signed 32-bit integer to represent 25-digit numbers, then you are probably in trouble.

Computers do *exactly* what you tell them to do. While the code you have written does not work exactly as you expect it to work, the computer does not really care what you expect of it. It simply executes your instructions. Making the interpretation of the instructions align with your expectations means that you have to change either the code or your expectations.

Let us call the problematic parts bugs. The bugs, whatever their nature, are still there, whether you like them or not. So how do you find them?

2 Debugging

You can have a look at the separate instruction on debugging and performance, which is available on the course home page.

You need to do two things to efficiently find your bugs; *verification* and *binary search.*

2.1 Verification

Verify that your code works correctly. Verify. Verify that intermediate results or data behaves as expected. If you are factoring a number, verify that the product of the computed factors does indeed equal the expected number. If you are producing solutions to an equation, verify that the solutions are in fact solutions. Verify that input to and output from every part of your program is reasonable and is utilized properly. Verify that you have chosen suitable data types. Verify everything that you can verify.

Often while programming, it is worth the time to take a few minutes to write some extra lines of code for verification. Catching run-time errors early can be very valuable (unless you really enjoy debugging).

2.2 Binary Search

The binary search part is just for locating the problem areas faster. Start troubleshooting (verifying) in the middle to decide if there is a problem in the first or last half. Repeat recursively.

And do remember that there can be more than one error.