9 Stress Tests

Unit tests help in assuring that eacvh component of a program performs the desired task. (Of course, a test only proves that for the given inputs the program produces the expected results.) But how do we know that the solution scales up to work in a production setting? One of the ways to assure this is through *stress tests*.

Stress tests run the program on large data sets and record the performance measurements of various kinds. We may worry about the time spent on specific tasks in the program, the variation in the time depending on the data set or some otehr circumstances, the amounf of memory needed, etc.

To introduce students to the idea that performance matters we ask the students to play detectives: their task is to match the timing results of several sorting algorithms with the appropriate algorithm. Furthermore, we ask them to reflect on the reasons why an algorithm behaves differently with different choices of how the data ordering is determined.

Start by running the student lab and trying to answer the questions we ask our students. Once you have done some measurements and thought about the questions, discuss your ideas with others.

In the second part we will look at the design of the program that generates the measurements. In this case we have one large data set available and our goal is a comparison of several different solutions to the same problem. Sometimes we may want to get the timing for different segments of a large program to identify the *hot spots*, where the optimization would be most desirable. The techinques we use in these examples can easily be adapted to such situations: we just record the CPU time in several places in our program.

However, in other circumstances, we may want to measure carefully the performance of just one algorithm with different inputs and data sets. Here the right approach is *randomized tests*. We cover this approach in the next lab.