Testing techniques

In the previous section, we've already done some testing and fixed some mistakes. In general, you should test your problem on the following groups of tests before submitting:

1. A few small manual tests.
2. A test for each possible type of answer (smallest answer, biggest answer, answer doesn't exist, etc.)
3. Test for time/memory limit: generate a test with the largest possible size of input ("max test"), run your program, measure time and memory consumption.
4. Tests for **corner cases**:

* Smallest possible "n": the length of the input sequence or string, the number of queries, etc.
* Equal numbers, equal letters in the string, more generally, equal objects in the input. Any two objects that are not restricted to be different in the problem statement can be equal.
* Largest numbers in the input allowed by the problem statement - for example, to test for integer overflow.
* Degenerate cases like empty set, three points on one line, a tree which consists of just one chain of nodes.

If after all of that you're sure that all answers are correct, but your solution is not accepted in the testing system, and you don't know what is the test case where your solution fails, there's the last resort called **stress testing** - a very efficient technique that will find you a test case for which your solution fails probably in 95% of cases not covered by the previously mentioned test types.

A stress test consists of four parts:

1. The solution you want to test.

2. A different, possibly trivial and very slow, but easy to implement and obviously correct solution to the problem.

3. A test generator.

4. An infinite loop in which a new test is generated, it is fed into both solutions, then the results are compared, and if they differ, the test and both answers are output, and the program stops, otherwise the loop repeats.

The idea behind stress testing is that if you have two correct solutions, and the answer to the problem is unique, then for any possible test case they are guaranteed to give the same answer. If, however, at least one of the solutions is incorrect, then with very high probability there exists a test on which their answers differ. The only case when it is not so is when there is a common mistake in both solutions, but that is very unlikely (unless the mistake is somewhere in the input/output routines which are common to both solutions - check for that separately). Indeed, if one solution is correct and the other is wrong, then there obviously exists a test case on which they differ. If both are wrong, but the bugs are different, then most probably there exists a test on which one solution gives a correct answer and another gives wrong answer, so they differ.

## Key take-aways

1. It is very important to write programs that work correctly on all the allowed inputs.
2. Testing is essential to writing correct programs.
3. First test on a few small manual tests, then test for each type of answer, then test on large test cases for time limit and memory limit, then test on corner cases.
4. After that, apply stress testing to ensure your program works - it will almost always lead to correct solution. You can do it before your first attempt to submit your solution - and will often get it right from the first attempt!
5. Stress testing consists of implementing the intended solution, another simple possible slow solution, a test generator and an infinite loop which generates tests and compares answers of the two solutions.
6. Always try to find the smallest test cases on which your solution fails.
7. Try different regions of the test space when generating cases for stress testing.

Here are the possible outcomes:

1. ***Good job!*** — Hurrah! Your solution passed, and you get a point!
2. ***Wrong answer.*** — Your solution has output incorrect answer for some test case. If it is a sample test case from the problem statement, or if you are solving Programming Assignment 1, you will also see the input data, the output of your program and the correct answer. Otherwise, you won't know the input, the output and the correct answer. Check that you consider all the cases correctly, avoid integer overflow, output the required whitespace, output the floating point numbers with the required precision, don't output anything in addition to what you are asked to output in the output specification of the problem statement. See this [reading](https://www.coursera.org/learn/algorithmic-toolbox/supplement/RULv4/stress-testing-the-almost-silver-bullet-for-debugging)on testing.
3. ***Time limit exceeded.*** — Your solution worked longer than the allowed time limit for some test case. If it is a sample test case from the problem statement, or if you are solving Programming Assignment 1, you will also see the input data and the correct answer. Otherwise, you won't know the input and the correct answer. Check again that your algorithm has a good enough running time estimate. Test your program locally on the test of maximum size allowed by the problem statement and see how long it takes. Check that your program doesn't wait for some input from the user which makes it wait forever. See this [reading](https://www.coursera.org/learn/algorithmic-toolbox/supplement/RULv4/stress-testing-the-almost-silver-bullet-for-debugging) on testing.
4. ***Memory limit exceeded.*** — Your solution used more than the allowed memory limit for some test case. If it is a sample test case from the problem statement, or if you are solving Programming Assignment 1, you will also see the input data and the correct answer. Otherwise, you won't know the input and the correct answer. Estimate the amount of memory that your program is going to use in the worst case and check that it is less than the memory limit. Check that you don't create too large arrays or data structures. Check that you don't create large arrays or lists or vectors consisting of empty arrays or empty strings, since those in some cases still eat up memory. Test your program locally on the test of maximum size allowed by the problem statement and look at its memory consumption in the system.
5. ***Cannot check answer. Perhaps output format is wrong.*** — This happens when you output something completely different than expected. For example, you are required to output word "Yes" or "No", but you output number 1 or 0, or vice versa. Or your program has empty output. Or your program outputs not only the correct answer, but also some additional information (this is not allowed, so please follow exactly the output format specified in the problem statement). Maybe your program doesn't output anything, because it crashes.
6. **Unknown signal 6**(or 7, or 8, or 11, or some other)**.** — This happens when your program crashes. It can be because of division by zero, accessing memory outside of the array bounds, using uninitialized variables, too deep recursion that triggers stack overflow, sorting with contradictory comparator, removing elements from an empty data structure, trying to allocate too much memory, and many other reasons. Look at your code and think about all those possibilities. Make sure that you use the same compilers and the same compiler options as we[do](https://www.coursera.org/learn/algorithmic-toolbox/supplement/rIqce/available-programming-languages). Try different testing techniques from this [reading](https://www.coursera.org/learn/algorithmic-toolbox/supplement/RULv4/stress-testing-the-almost-silver-bullet-for-debugging).
7. **Grading failed.** — Something very wrong happened with the system. Contact Coursera for help or write in the forums to let us know.
8. **Internal error: exception...** — Most probably, you submitted a compiled program instead of a source code.