CS 520
Homework 2
Delta Debugging

Due: **Thursday, November 15, 2018, 9:00 AM EST** via Moodle. You may work with others on this assignment but each student must submit his or her own write up, clearly specifying the collaborators. The write ups should be individual, not created jointly, and written in the student's own words. Late assignments will not be accepted without **prior** permission.

Overview

The goal of this assignment is, given a program P and a large test case T on which P fails for a particular reason, to use an automated technique called delta debugging to reduce T to a minimal test case which still cause P to fail for the same reason.

The assignment consists of:

1. Installing and using delta debugging to reduce a large test case that makes a sorting program fail to a small test case. Delta debugging is a perl script, and the sorting program is written in C. You'll need either a linux/unix-based operating system (MacOS works) or a virtual machine running such an OS.
2. Answering problem questions about delta debugging and the bug localization process.

Resources

The resources you need are all in a single git repository. You can download the resources by running
`git clone https://github.com/LASER-UMASS/DeltaDebuggingAssignment.git`
The repository contains the following files:

- `delta-2003.7.14.tar.gz`: Delta debugging — `http://www.st.cs.uni-saarland.de/dd` — is a fairly sophisticated program. We will be using a slightly older version, which has fewer features and is easier to use when you are just starting out.
- `mysort.c`: A buggy integer sorting program.
- `Makefile`: A makefile to build the sorting program.
- `big_failing_test.txt`: An input list of integers, one per line, on which the buggy integer sorting program fails.

Setup

1. Install the delta debugging tool by unpacking `delta-2003.7.14.tar.gz`. We will be using the program `delta/bin/delta` in the unpacked directory. It is a perl script, so no building is necessary. See `delta/README` after unpacking the distribution for more information.

2. Execute `make` and verify that `mysort` compiles. Its usage is:
   `mysort <input list> <timeout>`
   where:
   – `<input list>` is a text file containing a list of integers, one per line, to be sorted, and
   – `<timeout>` is a positive integer denoting the number of seconds the sorting routine should run.
   In the case of a timeout, `mysort` will print the message:
   `Error: timeout`
   The timeout is necessary because the sorting routine, being buggy, might loop indefinitely on some inputs. Ensure that `<timeout>` is reasonably large so that the program times out genuinely due to indefinite looping. The slower your machine, the higher should `<timeout>` be. Generally, a couple of seconds should suffice for most machines. Setting the timeout too high may mean that delta debugging takes a long time to reduce the test input.

Problem

The bug in the sorting function `sort` in `mysort.c` manifests when `mysort` is run with command line arguments consisting of a reasonably large timeout and the file `big_failing_test.txt`: it does not produce a sorted list of integers that is a permutation of the list of integers in `big_failing_test.txt`.

However, as its name suggests, `big_failing_test.txt` is not the minimal test case on which `mysort` fails. Use the delta debugging tool to obtain a minimal test case on which `mysort` fails and, moreover, for the same reason. You will need to **write** a script, called `script`, and run the command:

```
delta/bin/delta -test=script -cp_minimal=min_failing_test.txt < big_failing_test.txt
```

If script is written correctly, the above command will create a minimal test case named `min_failing_test.txt`. You might want to read the file `delta/doc/using_delta.txt` (except the sections "Topformflat" and "Multi-delta" since we are using line granularity and we are running the tool on a single file, `big_failing_test.txt`, instead of on multiple files).

Hint: Note that `delta` runs the test script in a subdirectory like `tmp0/arena/`. You may need to take this into account. Specifically, paths in the script should be relative to that subdirectory. For example, if you call `mysort` from the script, the path to `mysort` should start with `../../`

Deliverables

You should submit **3 files**:

- `script` and `min_failing_test.txt` files. We should be able to run the above command by placing your submitted files and our own copies of `mysort` and `big_failing_test.txt` in the same directory.
- Manually step through the sorting function `sort` in `mysort.c`, simulating its execution on the list of integers in `min_failing_test.txt`, and answer the following questions and submit either `writeup.pdf` or `writeup.txt` with the answers:

  1. Explain the cause of the bug in the source code of the `sort` function (that is, the reason why `mysort` fails on `min_failing_test.txt`).
  2. Characterize all input lists on which the sort function will fail because of the reason reported in (1) above.
  3. Report a fix for the bug.