In-class exercise 1

High-level goal

The high-level goal of this exercise is to learn about systematic debugging. This exercise has two parts, one about delta debugging and one about git bisect.

Set up

1. **Team up in groups of size 2 or 3 and assign yourself to a group on Canvas.**
2. Make sure that you have Apache Ant, a Java 8+ JDK, git, make, and cc installed.

Delta Debugging

1. Download and extract the following archive: *https://homes.cs.washington.edu/~rjust/courses/2021Winter/CSE503/dd.zip*
2. Run `make` to compile `mysort.c`.
3. Run `mysort passing.txt`.
4. Run `mysort failing.txt` (does not terminate; use ctrl-c to interrupt).

Background (story time)

A fellow developer hands you a legacy program, written in C. The program works most of the time, but your fellow developer ran into a problem with it. In the spirit of following best practices, they provide instructions with a concrete test case that shows the problem.

Unfortunately, the provided test case isn't quite minimal and your C skills are a little rusty. So, you decide to delta-debug (i.e., minimize) the test case first, before even thinking about the program itself. Luckily, there are plenty of delta-debugging implementations out there, so you "only" need to get it to work for this particular problem.

Now what?

1. Use delta debugging to minimize `failing.txt`. A wrapper script `dd.sh` for the `delta` command is provided, but you need to **write a `test` script** that executes mysort. Replace the `<your test oracle script>` in `dd.sh` with your own script. (You can write your script in any way and any programming language.) Note that the `delta` command executes your script in a subdirectory like tmp0/arena/. You may need to take this into account when working with relative paths and calling mysort from within your script. (Hints: How will your test script handle the fact that mysort may not terminate? What test outcome constitutes success for delta debugging?)

2. Using the minimized test case, debug and fix the issue in `mysort.c`.
   (Hint: you only got an archive; how can you still keep track of your changes?)

3. Manually create two test cases that trigger the problem in the original mysort and that have the following properties: both test cases have **four lines**; **test case 1** represents the **best case** scenario for delta debugging; **test case 2** represents the **worst case** scenario.

Git bisect

1. **Clone** the following **git repository** and read its README.md file: *https://bitbucket.org/rjust/basic-stats*
2. Test your set up by compiling, running, and testing the application. Note that **one test failure is expected** on the **most recent commit of master.**
3. Check out the version v1.0.0 (*git checkout v1.0.0*), and compile and test the application again. Note that **all tests are expected to pass** on this revision.

Background (story time)
The developers of this application followed reasonable development practices and used a test-driven approach until they published the first release. Everything just went haywire from there -- a lot of late-night, sleep-deprived, pizza-and-drinks hacking.

Unfortunately, the developers **introduced a defect** at some point **after** the first release (**v1.0.0**), which **has existed in the code since**. While the developers thought about and implemented some automated testing, their automated testing infrastructure does not catch the defect, and since they were so excited about coding, they never manually checked a single test run.
The **testing infrastructure** is a nightly run cron job, which executes the following command:
`ant clean test || reportBug`

This command builds the application from scratch and runs all tests using Ant. If `ant` fails, then the *`reportBug`* command notifies the developers via email; otherwise nothing happens.

Now what?
1. **Checkout** the most recent commit on **master** (*git checkout master*) and **run:** ***`ant clean test`***. Note the test failure and figure out **why the testing infrastructure** described above **does not catch this failure** (i.e., does not send an email).

2. Familiarize yourself with the version control history and **determine** the **number of commits between v1.0.0 and master (**current **HEAD** revision). Include v1.0.0 and HEAD when counting, and note the command(s) that you used.

3. Familiarize yourself with the git bisect command and use it to **identify** the **commit that introduced** the **defect between** version **v1.0.0** and **master**. Note the commit hash and log message of the defect-inducing commit. **Verify** that you **correctly identified** the defect-inducing commit. You may automate git bisect with a script.

Questions

Delta Debugging

1. What is the root cause of the bug in `mysort.c`?
2. Characterize **all** input lists on which `mysort.c` fails because of the reason in 1.
3. Provide a fix to the bug as unified diff (use `diff -u` or `git diff`).
4. Provide your two four-line test cases. Briefly explain for each test case, why it is the best case or worst case, respectively.

Git Bisect

5. Why does the described automated testing infrastructure not catch the defect?
6. How many commits exist between version v1.0.0 and master (including v1.0.0 and master)? What command(s) did you use to determine this number?
7. Which commit (commit hash and message) introduced the defect? How did you verify that this commit indeed introduced the defect?
8. After how many steps (git bisect calls) did you identify the defect-inducing commit?
9. What is the best-, worst-, and average-case run-time complexity of git bisect? Why?
10. Which git command can you generally use to undo a defect-inducing commit -- like the commit that you identified in this exercise? Can you undo the defect-inducing commit you identified using that git command? Briefly explain what problem may generally occur when undoing a commit and what best practices mitigate this problem.

Deliverables

A **plain-text file** with your answers to the questions above, including your bug fix as unified diff, your test script, and your four-line test cases. **Please list all group members.**