This MP covers some material from chapter 5 of the textbook and asks you to consider more test oracles for various situations (including grading some MP4 submissions). You will obtain some initial files and submit your added and modified files through SVN. Please do not add any unnecessary files (e.g., target). Your SVN directory is https://subversion.ews.illinois.edu/svn/sp16-cs498dm/${netid} as before.

There are four problems (and an easy, no problem task), worth a total of 120 points. As usual in this course, you need only 100 points to get the maximum score for this MP, i.e., if you have 100 points on each MP, you get A+ for the MP portion of the course.

You cannot collaborate on any problem for this MP! (You can and should collaborate with your team on your project, e.g., apply some grammar coverage or mutation testing on your project code!) You should not share your solutions or code on Piazza. If you use some online resource, you must credit the original source. Many online resources have bugs; if you find some bugs, please do report them.

No Problem [5 points]: If you notice some bugs related to the course (in book, slides, code, tests, text…) during this MP, please add a file mp5/bugs5.txt. You can also report bugs directly to Darko. These are the easiest points to get; please do report the bugs you find!

Problem 1 [45 points]: (*) Write a program that generates random tests in the JUnit format for a given class. For more details, search for TODO in SVN. Note that you can get some points for writing tests for this problem even if your code has problems. (If you want to test this using advanced Maven features, read about the generate-test-sources phase.) Please do NOT put your NetID or other personally identifiable information in the code or tests; we may share your work with the entire class.

Problem 2 [20 points]: (This is a modified version of Exercise 6 after Section 5.2.) For the method power, given in SVN, do the following:
(a) [8 points]: Define at least eight non-equivalent mutants for power. To create these mutants, use different operators from those effective mutation operators listed on pages 182-185, Chapter 5.
(b) [6 points]: Write a set of test inputs (not in JUnit format) that strongly kills all mutants.
(c) [6 points]: Define at least three equivalent mutants for power.
See SVN for details on writing your mutants and tests (search for TODO).

Problem 3 [20 points]: This problem comes in two variants: V1 and V2.
If you’re taking the course for 3 hours of credit, choose only one variant, V1 (x) or V2.
If you’re taking the course for 4 hours of credit, do both variants, V1 and V2

V1: John Micco discussed in his guest lecture how the Google TAP system finds “culprit” code changes that “broke” some test, i.e., changed the test outcome from “pass” to “fail”. Google has a lot of machine resources, so TAP is addressing this problem by checking many changes in parallel. Write your answers in mp5/prob3.txt.
(a) [10 points]: Describe the problem more precisely/technically and describe how TAP addresses it.
(b) [10 points]: Describe how you would address it if you wanted to take less machine time.
V2: Consider the following specification for a sort method:

```java
public static void sort(int[] s)
    // Precondition: s != null
    // Effects: sort elements of s (smaller elements first)
```

Suppose that sort is checked in the following way, with the code provided in SVN:

1. Make a copy t of s.
2. Apply the method sort to s.
3. Verify that every element in s is also an element in t.
4. Verify that every element in t is also an element in s.
5. Verify that s[i]<=s[i+1] for appropriate values of i.

(c) [5 points]: Do we need to test sort with s being null? What is an expected “output” in that case? Provide explanation for your answers; simple “yes” or “no” is not sufficient.

(d) [5 points]: The provided code that implements the checks has a few problems with array index bounds. Identify and correct those problems.

(e) [5 points]: Find initial and “sorted” values for s such that all of the given checks (with correct implementations for array index bounds) succeed, but the sorted version of s is still wrong. Propose an additional check or modify existing checks to detect this (counter) example with “sorted” values.

(f) [5 points]: Write a specification for the binarySearch method provided in SVN and implement a simple generic check for that method.

Problem 4 [30 points]: This problem asks you to grade tests for several MP4 submissions for Problem 1 on pairwise testing. The goal of this problem is to help you improve your skills on reading and evaluating tests, especially in cases where the oracles/assertions are not so easy to write. You can find the IDs of your assigned submissions in mp5/mp4-grade.csv, and you can find the actual submissions in https://subversion.ews.illinois.edu/svn/sp16-cs498dm/_shared/mp4-submissions. For each submission, you should assign a number of points between 0 and 10. You should also provide some comments if you give fewer than 10 points (or if you want to give some extra points in some special case!). Your evaluation can take several characteristics into account such as these: (1) do tests compile; (2) do tests pass; (3) is the test file readable; (4) are test names good; (5) is there duplication among the tests, and is it justified; (6) are the test inputs carefully selected; (7) are there enough tests; (8) are exceptional cases handled; (9) is the output checked in a proper way; (10) how does one submission compare to the others. Please feel free to ask on Piazza if you need some clarifications.