2262 BB Test Plan Review

**Introduction**

The purpose of this review is to offer feedback on the pros and cons of the test plan **2262** stored on the class repository for Dr. Roach’s Verification and Validation class.

**Analysis**

**Pros**

The 2262 test plan did well on having already implemented test cases to run with the Junit testing framework. To start, the test class titled appropriately *TableSorterTest* had the appropriate *setup()* and *teardown()* to have a *TableSorter* object supplied for testing the *TableSorter* class, and then having it cleaned up after use for the tests. The test class does well in that it first tests to verify the *TableSorter.isSorted(Table t)* method does indeed return the correct Boolean value (i.e. true or false). It goes through many cases from null pointers, empty tables, 1 x 1 matrices with positive, negative, and integer max and min values in varying combinations, as well as larger dimensional matrices such as an unsorted 300 x 300 matrix. The number of test cases performed is 18 unit test cases.

Following the testing of the *TableSorter.isSorted(Table t)* method, the tester is now (accepting risk of course) in a position to use the method as a form of verification for the *TableSorter.sortable(Table t)* method. This is done by using,

* *assertTrue( TableSorter.isSorted( TableSorter.sortable( Table t ) ) )*,
* *assertFalse( TableSorter.isSorted( TableSorter.sortable( Table t ) ) ),*

in order to verify if indeed the table is sorted or not. Reason for using this approach, is due to the fact that a table could be sorted many different ways, as defined by the criteria of the assignment,

*”The matrix is considered sorted if each column individually is sorted and each row individually is sorted. It is possible to have more than one sorting for some matrices.” –* ***\*Piazza “question on sorting assignment” Post\****

Therefore, the tester must rely on the previously tested *TableSorter.isSorted( Table t )* method to verify the table is sorted (accepting the risk of doing so).

The test class does well in keeping with Junit formatting standards in that the code is written in a readable format for those familiar with Junit. It uses the appropriate Junit assertion methods to test the methods being tested within the *TableSorter* class.

**Cons**

A con to this test plan is that it has the test cases implemented, but if a tester would like to test other aspects of the input/output nature of the class (i.e. black-box testing) they do not have a description of doing so and what to expect. The tester would have to rely on their expertise which necessarily is not a bad thing. Considering the fact that the test cases are written in a readable format a tester could hypothetically use a “copy-paste” style of writing tests to test out other variations not present.

The test matrices only span 1 x 1, 5 x 5, 4 x 4, 3 x 3, and 300 x 300 with none for 2 x 2 which should not be a problem, but leaves a tester wondering why skip such a case. These matrices are used for the testing of *TableSorter.isSorted( Table t )* method. The values used for the matrices are mostly positive with one test case having all negative and a zero value. Some tests on a mixture of positive and negative values should be tested with varying percentages, maybe 10/90, 20/80, 30/70, 60/40, and 50/50. This could help verify that the *TableSorter.isSorted( Table t )* method is capable of handling the mixture of values with varied percentages.

The test plan has test cases that tests two tables one that is not sorted at all and has a column-wise tie. The other tests a table with row-wise tie and is partially-sorted, but does not explain what this necessarily means, because it is not obvious why the test is happening. Specifically the terms row-wise tie and column-wise tie are not adequately defined.