Test plan

Version 1.4

2-12-2020

Document Control

Approval

The Guidance Team and the customer shall approve this document.

Document Change Control

|  |  |
| --- | --- |
| Initial Release: | 1.0 |
| Current Release: | 1.4 |
| Indicator of Last Page in Document: | $ |
| Date of Last Review: | 2-12-2020 |
| Date of Next Review: | 2-12-2020 |
| Target Date for Next Update: | 2-12-2020 |

Distribution List

This following list of people shall receive a copy of this document every time a new version of this document becomes available:

Professor:

Dr. Steve Roach

Software Team Member:

Change Summary

The following table details changes made between versions of this document

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Modifier | Description |
| 1.0 | 2-10-2020 |  | Creation of document. |
| 1.1 | 2-10-2020 |  | Introduction |
| 1.2 | 2-10-2020 |  | Section 2 |
| 1.3 | 2-11-2020 |  | Section 3 |
| 1.4 | 2-12-2020 |  | Section 4 |

Note: The template presented in this document was taken from:

Donaldson, S., and S. Siegel, *Successful Software Development*. Upper Saddle River, NJ: Prentice Hall, 2001, pp. 321-323.

Note: The template presented in this document was taken from: Donaldson, S., and S. Siegel, *Successful Software Development*. Upper Saddle River, NJ: Prentice Hall, 2001, pp. 321-323 and modified by Humberto Mendoza and Steve Roach.

Supplementary information is from:

Pfleeger, S. *Software Engineering, Theory and Practice*. Upper Saddle River, NJ: Prentice Hall, 1998, p. 365.

Table of Contents

[Document Control ii](#_Toc22915465)

[Approval ii](#_Toc22915466)

[Document Change Control ii](#_Toc22915467)

[Distribution List ii](#_Toc22915468)

[Change Summary ii](#_Toc22915469)

[1. Introduction 1](#_Toc22915470)

[1.1. Purpose 1](#_Toc22915471)

[1.2. Scope 1](#_Toc22915472)

[1.3. System Overview 1](#_Toc22915473)

[1.4. Suspension and Exit Criteria 1](#_Toc22915474)

[1.5. Document Overview 1](#_Toc22915475)

[1.6. References 2](#_Toc22915476)

[2. Testing Approach 3](#_Toc22915478)

[3. Test issorted 5](#_Toc22915479)

[3.1. Test is1 5](#_Toc22915480)

[3.2. Test is2 8](#_Toc22915480)

[3.3. Test is3 10](#_Toc22915480)

[3.4. Test is4 12](#_Toc22915480)

[3.5. Test is5 14](#_Toc22915480)

[4. Test Sortable 18](#_Toc22915479)

[4.1. Test s1 18](#_Toc22915480)

[4.2. Test s2 21](#_Toc22915480)

[4.3. Test s3 23](#_Toc22915480)

[4.4. Test s4 26](#_Toc22915480)

# Introduction

## Purpose

The purpose of this document is to provide a test plan that describes the overall strategy for testing the Table Sorter programming assignment. The intended audience for this document is for me to use as a testing reference when implementing the Table Sorter class. Thus, the test plan will ensure that I provide Dr. Roach with a software program that has been tested for bugs and discrepancies and ensures a high degree of robustness. This test plan ensures that I have verified and validated the Table Sorter class.

## Scope

In order to test the Table Sorter class I will identify the testing strategy to be use for the Table Sorter class and use the test plan to verify and validate such program in order to further improve the reliability of Table Sorter.

## System Overview

The testing approach was to create a test case for each requirement that was provided. Moreover, a test suite describes all the test cases by diving them into tables defined by the characteristics they share in common. Thus, the overall system overview provides a series of test suites that test the functionality of the isSorted (iS) and Sortable (S) methods from the Table Sorter class.

## Suspension and Exit Criteria

The suspension criteria to be followed in the test plan is to stop the testing phase whenever any critical test case fails to meet the requirements or when 50% of the non-critical test cases fail at any given time. Furthermore, the exit criteria for finishing the testing phase is defined by the following statements. All critical tests must successfully pass the test cases and 85% of the non-critical tests must pass for the testing phase to conclude. Thus, I will ensure that I have successfully verified and validated the Table Sorter class.

## Document Overview

The Test Plan document is composed of 4 major sections. The document control and section 1 (Introduction) provide the overview and introductory information of the Test Plan. Furthermore, section 1 includes 6 subsections which include 1.1 Purpose, 1.2 Scope, 1.3 System Overview, 1.4 Suspension and Exit Criteria, 1.5 Document Overview, and 1.6 References.

Section 2 provides the testing approach strategy by describing a set of test suites of BEAT.

Section 3 provides the test plan with a set of test cases in order to document test procedures, establish test methods, and describe each test case in detail.

Section 4 provides the test plan with a set of test cases in order to document test procedures, establish test methods, and describe each test case in detail.

## References

[1] A. Balderas et al, Behavior Extraction and Analysis Tool Test Plan v1.6, UTEP, 2019

[2] Adam, Anju, R. Singh, S. Aslam, Audrey, Alloch, Akoll, Hanif, S. Mathew, Ayo, Deepan, Dolly, Jitender, M. Raja, J. D, Gitesh, Katya, Ali, Aditya Aggarwal, Evergreen, and Sheetal, “Test Plan Tutorial: A Guide To Write A Software Test Plan Document From Scratch,” *Software Testing Help*, 04-Dec-2019. [Online]. Available: https://www.softwaretestinghelp.com/how-to-write-test-plan-document-software-testing-training-day3/. [Accessed: 10-Feb-2020].

[3] Fred, Sabari, Andy, Alex, K. C, and Samera, “Difference Between Test Plan, Test Strategy, Test Case, and Test Scenario,” *Software Testing Help*, 31-Dec-2019. [Online]. Available: https://www.softwaretestinghelp.com/difference-between-test-plan-test-strategy-test-case-test-script-test-scenario-and-test-condition/. [Accessed: 10-Feb-2020].

[4] N. Ahmad, Venkatesh, Aniket, Prasanna, Ajay, S. Nohanty, Rakesh, Omar, Jane, N. D. Talele, P. Yelsangi, A. Khan, A. Shiny, G. Gaikwad, Alok, Samera, Shiva Balaji, and Ars, “How to Write Test Cases: The Ultimate Guide with Examples,” *Software Testing Help*, 08-Jan-2020. [Online]. Available: https://www.softwaretestinghelp.com/how-to-write-effective-test-cases-test-cases-procedures-and-definitions/. [Accessed: 10-Feb-2020].

# Testing Approach

The Test Plan provides a testing approach strategy describe in table 1 and 2. It provides a description of the testing suite along with test cases pertaining to that specific test suite. Furthermore, an objective of each test case is given stating the purpose of the test case. The criticality column provides a rating of how important a test case is to the Table Sorter class. There are three categories of criticality. Critical, meaning that the class proper functionality is interfered with. Normal, does not interfere with core functionality. Lastly, low, which does not impact the Table Sorter class functionality. I assume that a I must only test the Table Sorter class and that any other type of test that relates to the Table class such as test cases related to invalid input are to be created in another test plan pertaining to the Table class.

Table 1: Test Plan for Test Suite isSorted

|  |  |  |
| --- | --- | --- |
| **TEST SUITE iSorted** | | |
| Description of Test Suite | This suite focuses on the *isSorted* instance method along with any other functions that are related to its implementation. | |
| **Test Case Identifier** | **Objective** | **Criticality** |
| iS1 | Verify and validate that the *isSorted* method(s) returns True when Table *t* is row wise and column wise sorted in ascending order. | Critical |
| iS2 | Verify and validate that the *isSorted* method(s) returns False when Table *t* is not row wise and column wise sorted in ascending order. | Critical |
| iS3 | Verify and validate that *isSorted* method(s) does not return anything when Table t is null. | Normal |
| iS4 | Verify and validate that the *isSorted* method(s) returns False when Table *t* is row wise sorted in ascending order, but not column wise. | Normal |
| iS5 | Verify and validate that the *isSorted* method(s) returns False when Table *t* is column wise sorted in ascending order, but not row wise. | Normal |

Table 2: Test Plan for Test Suite Sortable

|  |  |  |
| --- | --- | --- |
| **TEST SUITE Sortable** | | |
| Description of Test Suite | This suite focuses on the *Sortable* instance method along with any other functions that are related to its implementation. | |
| **Test Case Identifier** | **Objective** | **Criticality** |
| S1 | Verify and validate that the *sortable* method(s) sorts Table *t* if it is not sorted row wise and column wise in ascending order. | Critical |
| S2 | Verify and validate that the *sortable* method(s) does not sort Table *t* row wise only, but also column wise. | Normal |
| S3 | Verify and validate that the *sortable* method(s) does not sort Table *t* column wise only, but also row wise. | Normal |
| S4 | Verify and validate that the *sortable* method(s) does not sort row wise and column wise in ascending order when Table *t* is null. | Normal |

# Test isSorted

This section documents the testing procedures, and the testing inputs and outputs. It establishes the testing methods and describes each test case in detail.

## Test iS1

**Objective:** To verify that the *isSorted* method(s) returns True if Table *t* is row wise and column wise sorted in ascending order.

**Notes:** The iS1 test case must last anywhere between 30 seconds to 60 seconds for completion of the mechanism. The requirements are that a valid *NxN* list of integers is passed as a table of values.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test No.: iS1 | | | | Current Status: Pending | | |
| Test title: True is returned when a valid *NxN* list of integers is passed as input. | | | | | | |
| Testing approach: This test will be conducted using a Java compiler in the command prompt and a text editor or an IDE. Behavior is observed when output is returned into the screen. | | | | | | |
| STEP  1 | OPERATOR ACTION  Begin testing by opening the Table Sorter class. | PURPOSE  Initiate the Table Sorter class. | | | EXEPCTED RESULTS  The Table Sorter class is opened in a text editor or IDE. | COMMENTS  N/A |
| 2 | On the Table Sorter class, navigate to the *main* method and find where the Table object is instantiated. | Find where the *main* method is located, and the Table object is instantiated. | | | The *main* method has been found and a Table object has been created. | A Table object shall be created before calling the *isSorted* method(s).  Ex*: Table table = new Table();* |
| 2 | Find the *GetTable* method and provide the following directory path *“C:\Users\username\Documents\table.txt”* that opens the *“table.txt”* file. Save the Table Sorter class.  Ex *table.txt*: {1,2,3,4,5,6,7,8,9} | Provide a valid *NxN* list of integers of a table of values. | | | A valid *NxN* list of integers is passed as input to the Table Sorter class. | The list of integers is row wise and column wise sorted in ascending order.  Note: Move to Alt. Step 2.1 if the class does not use the *GetTable* method which receives a text file as input. |
| 3 | Find where the *isSorted* method(s) is located. | Locate where the *isSorted* method receives as input the *NxN* list of integers as a Table object. | | | The *isSorted* method has been located. | N/A |
| 4 | Compile and run the Table Sorter class. Either by using the command prompt or an IDE. | The Table Sorter class has been executed. *isSorted* returns True. | | | An output is given in the screen that reads “True”. | If using the command prompt to compile execute: *javac filename.java*  To run: *java filename*  If using an IDE run by clicking on the run button. |
| Alt. Step  2.1 | Find the *Table* method and provide the *N* and valid array list of integers.  Ex: [9, {1,2,3,4,5,6,7,8,9}] | Provide a square *N* and a valid list of integers of a table of values. | | | A valid *NxN* list of integers is passed as input to the Table Sorter class. | The list of integers is row wise and column wise sorted in ascending order.  Note: Move back to Step 3 |
| Concluding Remarks: The *isSorted* method(s) returns True from a valid *NxN* list of integers that is row wise and column wise sorted in ascending order. | | | | | | |
| Testing Team: | | | Date Completed:  TBD | | | |

## Test iS2

**Objective:** To verify that the *isSorted* method(s) returns False if a table is not row wise and column wise sorted in ascending order.

**Notes:** The iS2 test case must last anywhere between 30 seconds to 60 seconds for completion of the mechanism.

The requirements are that a valid *NxN* list of integers is passed as a table of values.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test No.: iS1 | | | | Current Status: Pending | | |
| Test title: False is returned when a valid *NxN* list of integers is passed as input. | | | | | | |
| Testing approach: This test will be conducted using a Java compiler in the command prompt or an IDE. Behavior is observed when output is returned into the screen. | | | | | | |
| STEP  1 | OPERATOR ACTION  Begin testing by opening the Table Sorter class. | PURPOSE  Initiate the Table Sorter class. | | | EXEPCTED RESULTS  The Table Sorter class is ready to receive input. | COMMENTS  N/A |
| 2 | Find the *GetTable* method and provide the following directory path *“C:\Users\username\Documents\table.txt”* that opens the *“table.txt”* file. Save the Table Sorter class.  Ex *table.txt*: {2,3,4,6,8,1,5,9,7} | Provide a valid *NxN* list of integers of a table of values. | | | A valid *NxN* list of integers is passed as input to the Table Sorter class. | The list of integers is not row wise and column wise sorted in ascending order.  Note: Move to Alt. Step 2.1 if the class does not use the *GetTable* method which receives a text file as input. |
| 3 | Find where the *isSorted* method(s) is located. | Locate where the *isSorted* method receives as input the *NxN* list of integers as a Table object. | | | The *isSorted* method has been located | N/A |
| 4 | Compile and run the Table Sorter class. Either by using the command prompt or an IDE. | The Table Sorter class has been executed. *isSorted* returns False. | | | An output is given in the screen that reads “False”. | If using the command prompt to compile execute: *javac filename.java*  To run: *java filename*  If using an IDE run by clicking on the run button. |
| Alt. Step  2.1 | Find the *Table* method and provide the *N* and valid array list of integers.  Ex: [9, {2,3,4,6,8,1,5,9,7}] | Provide a square *N* and a valid list of integers of a table of values. | | | A valid *NxN* list of integers is passed as input to the Table Sorter class. | The list of integers is not row wise and column wise sorted in ascending order.  Note: Move back to Step 3 |
| Concluding Remarks: The *isSorted* method(s) returns False from a valid *NxN* list of integers that is not row wise and column wise sorted in ascending order. | | | | | | |
| Testing Team: | | | Date Completed:  TBD | | | |

## Test iS3

**Objective:** To verify that the *isSorted* method(s) does not return anything when Table *t* is null.

**Notes:** The iS2 test case must last anywhere between 30 seconds to 60 seconds for completion of the mechanism.

The requirements are nothing is given as input to the Table Sorter program.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test No.: iS3 | | | | Current Status: Pending | | |
| Test title: Nothing is returned when a null list of integers is passed as input. | | | | | | |
| Testing approach: This test will be conducted using a Java compiler in the command prompt or an IDE. Behavior is observed when output is returned into the screen. | | | | | | |
| STEP  1 | OPERATOR ACTION  Begin testing by opening the Table Sorter class. | PURPOSE  Initiate the Table Sorter class. | | | EXEPCTED RESULTS  The Table Sorter class is ready to receive input. | COMMENTS  N/A |
| 2 | Find the *GetTable* method and provide the following directory path *“C:\Users\username\Documents\table.txt”* that opens the *“table.txt”* file. Save the Table Sorter class.  Ex *table.txt*: {} | Provide a null list of integers. | | | A null list of integers is passed as input to the Table Sorter class. | Note: Move to Alt. Step 2.1 if the class does not use the *GetTable* method which receives a text file as input. |
| 3 | Find where the *isSorted* method(s) is located. | Locate where the *isSorted* method receives as input a null list of integers as a Table object. | | | The *isSorted* method has been located | N/A |
| 4 | Compile and run the Table Sorter class. Either by using the command prompt or an IDE. | The Table Sorter class has been executed. *isSorted* returns nothing. | | | An output is given in the screen that reads “”. | If using the command prompt to compile execute: *javac filename.java*  To run: *java filename*  If using an IDE run by clicking on the run button. |
| Alt. Step  2.1 | Find the *Table* method and provide the N and null array list of integers.  Ex: [0, {}] | Provide a square *N* and a null list of integers of a table of values. | | | A null list of integers is passed as input to the Table Sorter class. | Note: Move back to Step 3 |
| Concluding Remarks: The *isSorted* method(s) returns nothing from null list of integers. | | | | | | |
| Testing Team: | | | Date Completed:  TBD | | | |

## Test iS4

**Objective:** To verify that the *isSorted* method(s) returns False if a table is row wise sorted in ascending order, but not column wise.

**Notes:** The iS2 test case must last anywhere between 30 seconds to 60 seconds for completion of the mechanism.

The requirements are that a valid *NxN* list of integers is passed as a table of values.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test No.: iS1 | | | | Current Status: Pending | | |
| Test title: False is returned when a valid *NxN* list of integers that is sorted row wise in ascending order, but not column wise is passed as input. | | | | | | |
| Testing approach: This test will be conducted using a Java compiler in the command prompt or an IDE. Behavior is observed when output is returned into the screen. | | | | | | |
| STEP  1 | OPERATOR ACTION  Begin testing by opening the Table Sorter class. | PURPOSE  Initiate the Table Sorter class. | | | EXEPCTED RESULTS  The Table Sorter class is ready to receive input. | COMMENTS  N/A |
| 2 | Find the *GetTable* method and provide the following directory path *“C:\Users\username\Documents\table.txt”* that opens the *“table.txt”* file. Save the Table Sorter class.  Ex *table.txt*: {1,3,4,5,6,8,2,7,9} | Provide a valid *NxN* list of integers of a table of values. | | | A valid *NxN* list of integers is passed as input to the Table Sorter class. | The list of integers is row wise sorted in ascending order, but not column wise.  Note: Move to Alt. Step 2.1 if the class does not use the *GetTable* method which receives a text file as input. |
| 3 | Find where the *isSorted* method(s) is located. | Locate where the *isSorted* method receives as input the *NxN* list of integers as a Table object. | | | The *isSorted* method has been located | N/A |
| 4 | Compile and run the Table Sorter class. Either by using the command prompt or an IDE. | The Table Sorter class has been executed. *isSorted* returns False. | | | An output is given in the screen that reads “False”. | If using the command prompt to compile execute: *javac filename.java*  To run: *java filename*  If using an IDE run by clicking on the run button. |
| Alt. Step  2.1 | Find the *Table* method and provide the N and valid array list of integers.  Ex: [9, {1,3,4,5,6,8,2,7,9}] | Provide a square *N* and a valid list of integers of a table of values. | | | A valid *NxN* list of integers is passed as input to the Table Sorter class. | The list of integers is row wise sorted in ascending order, but not column wise.  Note: Move back to Step 3 |
| Concluding Remarks: The *isSorted* method(s) returns False from a valid *NxN* list of integers that is row wise sorted in ascending order, but not column wise. | | | | | | |
| Testing Team: | | | Date Completed:  TBD | | | |

## Test iS5

**Objective:** To verify that the *isSorted* method(s) returns False if a table is column wise sorted in ascending order, but not row wise.

**Notes:** The iS2 test case must last anywhere between 30 seconds to 60 seconds for completion of the mechanism. The requirements are that a valid *NxN* list of integers is passed as a table of values.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test No.: iS1 | | | | Current Status: Pending | | |
| Test title: False is returned when a valid *NxN* list of integers that is sorted column wise in ascending order, but not row wise is passed as input. | | | | | | |
| Testing approach: This test will be conducted using a Java compiler in the command prompt or an IDE. Behavior is observed when output is returned into the screen. | | | | | | |
| STEP  1 | OPERATOR ACTION  Begin testing by opening the Table Sorter class. | PURPOSE  Initiate the Table Sorter class. | | | EXEPCTED RESULTS  The Table Sorter class is ready to receive input. | COMMENTS  N/A |
| 2 | Find the *GetTable* method and provide the following directory path *“C:\Users\username\Documents\table.txt”* that opens the *“table.txt”* file. Save the Table Sorter class.  Ex *table.txt*: {1,4,3,2,5,6,7,8,9} | Provide a valid *NxN* list of integers of a table of values. | | | A valid *NxN* list of integers is passed as input to the Table Sorter class. | The list of integers is column wise sorted in ascending order, but not row wise.  Note: Move to Alt. Step 2.1 if the class does not use the *GetTable* method which receives a text file as input. |
| 3 | Find where the *isSorted* method(s) is located. | Locate where the *isSorted* method receives as input the *NxN* list of integers as a Table object. | | | The *isSorted* method has been located | N/A |
| 4 | Compile and run the Table Sorter class. Either by using the command prompt or an IDE. | The Table Sorter class has been executed. *isSorted* returns False. | | | An output is given in the screen that reads “False”. | If using the command prompt to compile execute: *javac filename.java*  To run: *java filename*  If using an IDE run by clicking on the run button. |
| Alt. Step  2.1 | Find the *Table* method and provide the N and valid array list of integers.  Ex: [9, {1,4,3,2,5,6,7,8,9}] | Provide a square *N* and a valid list of integers of a table of values. | | | A valid *NxN* list of integers is passed as input to the Table Sorter class. | The list of integers is column wise sorted in ascending order, but not row wise.  Note: Move back to Step 3 |
| Concluding Remarks: The *isSorted* method(s) returns False from a valid *NxN* list of integers that is column wise sorted in ascending order, but not row wise. | | | | | | |
| Testing Team: | | | Date Completed:  TBD | | | |

# Test Sortable

This section documents the testing procedures, and the testing inputs and outputs. It establishes the testing methods and describes each test case in detail.

## Test S1

**Objective:** To verify that the *Sortable* method(s) sorts Table *t* if it is not row wise and column wise sorted in ascending order.

**Notes:** The S1 test case must last anywhere between 30 seconds to 60 seconds for completion of the mechanism. The requirements are that a valid *NxN* list of integers is passed as a table of values.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test No.: S1 | | | | Current Status: Pending | | |
| Test title: Sorts Table *t* row wise and column wise in ascending order when a valid *NxN* list of integers is passed as input. | | | | | | |
| Testing approach: This test will be conducted using a Java compiler in the command prompt and a text editor or an IDE. Behavior is observed when output is returned into the screen. | | | | | | |
| STEP  1 | OPERATOR ACTION  Begin testing by opening the Table Sorter class. | PURPOSE  Initiate the Table Sorter class. | | | EXEPCTED RESULTS  The Table Sorter class is opened in a text editor or IDE. | COMMENTS  N/A |
| 2 | On the Table Sorter class, navigate to the *main* method and find where the Table object is instantiated. | Find where the *main* method is located, and the Table object is instantiated. | | | The *main* method has been found and a Table object has been created. | A Table object shall be created before calling the *sortable* method(s).  Ex*: Table table = new Table();* |
| 2 | Find the *GetTable* method and provide the following directory path *“C:\Users\username\Documents\table.txt”* that opens the *“table.txt”* file. Save the Table Sorter class.  Ex *table.txt*: {2,3,4,6,8,1,5,9,7} | Provide a valid *NxN* list of integers of a table of values. | | | A valid *NxN* list of integers is passed as input to the Table Sorter class. | The list of integers is not row wise and column wise sorted in ascending order.  Note: Move to Alt. Step 2.1 if the class does not use the *GetTable* method which receives a text file as input. |
| 3 | Find where the *sortable* method(s) is located. | Locate where the *sortable* method receives as input the *NxN* list of integers as a Table object. | | | The *sortable* method has been located. | N/A |
| 4 | Compile and run the Table Sorter class. Either by using the command prompt or an IDE. | The Table Sorter class has been executed. *sortable* returns a sorted array in ascending order. | | | An output is given in the screen.  Ex: “1 2 3  4 5 6  7 8 9” | If using the command prompt to compile execute: *javac filename.java*  To run: *java filename*  If using an IDE run by clicking on the run button. |
| Alt. Step  2.1 | Find the *Table* method and provide the *N* and valid array list of integers.  Ex: [9, {2,3,4,6,8,1,5,9,7}] | Provide a square *N* and a valid list of integers of a table of values. | | | A valid *NxN* list of integers is passed as input to the Table Sorter class. | The list of integers is not row wise and column wise sorted in ascending order.  Note: Move back to Step 3 |
| Concluding Remarks: The *sortable* method(s) sorts Table *t* from a valid *NxN* list of integers that is not row wise and column wise sorted in ascending order. | | | | | | |
| Testing Team: | | | Date Completed:  TBD | | | |

## Test S2

**Objective:** To verify the *Sortable* method(s) does not sort Table *t* row wise only, but also column wise.

**Notes:** The S2 test case must last anywhere between 30 seconds to 60 seconds for completion of the mechanism. The requirements are that a valid *NxN* list of integers is passed as a table of values.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test No.: S2 | | | | Current Status: Pending | | |
| Test title: Table *t* is not only sorted row wise in ascending order, but also column wise when a valid *NxN* list of integers is passed as input. | | | | | | |
| Testing approach: This test will be conducted using a Java compiler in the command prompt and a text editor or an IDE. Behavior is observed when output is returned into the screen. | | | | | | |
| STEP  1 | OPERATOR ACTION  Begin testing by opening the Table Sorter class. | PURPOSE  Initiate the Table Sorter class. | | | EXEPCTED RESULTS  The Table Sorter class is opened in a text editor or IDE. | COMMENTS  N/A |
| 2 | On the Table Sorter class, navigate to the *main* method and find where the Table object is instantiated. | Find where the *main* method is located, and the Table object is instantiated. | | | The *main* method has been found and a Table object has been created. | A Table object shall be created before calling the *sortable* method(s).  Ex*: Table table = new Table();* |
| 2 | Find the *GetTable* method and provide the following directory path *“C:\Users\username\Documents\table.txt”* that opens the *“table.txt”* file. Save the Table Sorter class.  Ex *table.txt*: {1,3,4,5,6,8,2,7,9} | Provide a valid *NxN* list of integers of a table of values. | | | A valid *NxN* list of integers is passed as input to the Table Sorter class. | The list of integers is sorted row wise in ascending order, but not column wise.  Note: Move to Alt. Step 2.1 if the class does not use the *GetTable* method which receives a text file as input. |
| 3 | Find where the *sortable* method(s) is located. | Locate where the *sortable* method receives as input the *NxN* list of integers as a Table object. | | | The *sortable* method has been located. | N/A |
| 4 | Compile and run the Table Sorter class. Either by using the command prompt or an IDE. | The Table Sorter class has been executed. *sortable* returns a sorted array in ascending order. | | | An output is given in the screen.  Ex: “1 2 3  4 5 6  7 8 9” | If using the command prompt to compile execute: *javac filename.java*  To run: *java filename*  If using an IDE run by clicking on the run button. |
| Alt. Step  2.1 | Find the *Table* method and provide the *N* and valid array list of integers.  Ex: [9, {1,3,4,5,6,8,2,7,9}] | Provide a square *N* and a valid list of integers of a table of values. | | | A valid *NxN* list of integers is passed as input to the Table Sorter class. | The list of integers is row wise sorted in ascending order, but not column wise.  Note: Move back to Step 3 |
| Concluding Remarks: The *sortable* method(s) sorts Table *t* from a valid *NxN* list of integers not only row wise but also column wise. | | | | | | |
| Testing Team: | | | Date Completed:  TBD | | | |

## Test S3

**Objective:** To verify the *Sortable* method(s) does not sort Table *t* column wise only, but also row wise.

**Notes:** The S3 test case must last anywhere between 30 seconds to 60 seconds for completion of the mechanism. The requirements are that a valid *NxN* list of integers is passed as a table of values.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test No.: S3 | | | | Current Status: Pending | | |
| Test title: Table *t* is not only sorted column wise in ascending order, but also row wise when a valid *NxN* list of integers is passed as input. | | | | | | |
| Testing approach: This test will be conducted using a Java compiler in the command prompt and a text editor or an IDE. Behavior is observed when output is returned into the screen. | | | | | | |
| STEP  1 | OPERATOR ACTION  Begin testing by opening the Table Sorter class. | PURPOSE  Initiate the Table Sorter class. | | | EXEPCTED RESULTS  The Table Sorter class is opened in a text editor or IDE. | COMMENTS  N/A |
| 2 | On the Table Sorter class, navigate to the *main* method and find where the Table object is instantiated. | Find where the *main* method is located, and the Table object is instantiated. | | | The *main* method has been found and a Table object has been created. | A Table object shall be created before calling the *sortable* method(s).  Ex*: Table table = new Table();* |
| 2 | Find the *GetTable* method and provide the following directory path *“C:\Users\username\Documents\table.txt”* that opens the *“table.txt”* file. Save the Table Sorter class.  Ex *table.txt*: {1,4,3,2,5,6,7,8,9} | Provide a valid *NxN* list of integers of a table of values. | | | A valid *NxN* list of integers is passed as input to the Table Sorter class. | The list of integers is sorted column wise in ascending order, but not row wise.  Note: Move to Alt. Step 2.1 if the class does not use the *GetTable* method which receives a text file as input. |
| 3 | Find where the *sortable* method(s) is located. | Locate where the *sortable* method receives as input the *NxN* list of integers as a Table object. | | | The *sortable* method has been located. | N/A |
| 4 | Compile and run the Table Sorter class. Either by using the command prompt or an IDE. | The Table Sorter class has been executed. *sortable* returns a sorted array in ascending order. | | | An output is given in the screen.  Ex: “1 2 3  4 5 6  7 8 9” | If using the command prompt to compile execute: *javac filename.java*  To run: *java filename*  If using an IDE run by clicking on the run button. |
| Alt. Step  2.1 | Find the *Table* method and provide the *N* and valid array list of integers.  Ex: [9, {1,4,3,2,5,6,7,8,9}] | Provide a square *N* and a valid list of integers of a table of values. | | | A valid *NxN* list of integers is passed as input to the Table Sorter class. | The list of integers is column wise sorted in ascending order, but not row wise.  Note: Move back to Step 3 |
| Concluding Remarks: The *sortable* method(s) sorts Table *t* from a valid *NxN* list of integers not only column wise but also row wise. | | | | | | |
| Testing Team: | | | Date Completed:  TBD | | | |

## S4

**Objective:** To verify the *Sortable* method(s) does not sort Table *t* row wise andcolumn wise in ascending order when *t* is null.

**Notes:** The S4 test case must last anywhere between 30 seconds to 60 seconds for completion of the mechanism. The requirements are that a null list of integers is passed as a table of values.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test No.: S4 | | | | Current Status: Pending | | |
| Test title: Table *t* is not sorted row wise and column wise in ascending order when *t* is null. | | | | | | |
| Testing approach: This test will be conducted using a Java compiler in the command prompt and a text editor or an IDE. Behavior is observed when output is returned into the screen. | | | | | | |
| STEP  1 | OPERATOR ACTION  Begin testing by opening the Table Sorter class. | PURPOSE  Initiate the Table Sorter class. | | | EXEPCTED RESULTS  The Table Sorter class is opened in a text editor or IDE. | COMMENTS  N/A |
| 2 | On the Table Sorter class, navigate to the *main* method and find where the Table object is instantiated. | Find where the *main* method is located, and the Table object is instantiated. | | | The *main* method has been found and a Table object has been created. | A Table object shall be created before calling the *sortable* method(s).  Ex*: Table table = new Table();* |
| 2 | Find the *GetTable* method and provide the following directory path *“C:\Users\username\Documents\table.txt”* that opens the *“table.txt”* file. Save the Table Sorter class.  Ex *table.txt*: {} | Provide a null list of integers of a table of values. | | | A null list of integers is passed as input to the Table Sorter class. | Note: Move to Alt. Step 2.1 if the class does not use the *GetTable* method which receives a text file as input. |
| 3 | Find where the *sortable* method(s) is located. | Locate where the *sortable* method receives as input the null list of integers as a Table object. | | | The *sortable* method has been located. | N/A |
| 4 | Compile and run the Table Sorter class. Either by using the command prompt or an IDE. | The Table Sorter class has been executed. *sortable* returns nothing. | | | An output is given in the screen.  Ex: “” | If using the command prompt to compile execute: *javac filename.java*  To run: *java filename*  If using an IDE run by clicking on the run button. |
| Alt. Step  2.1 | Find the *Table* method and provide the *N* and valid array list of integers.  Ex: [0, {}] | Provide a square *N* and a null list of integers of a table of values. | | | A null list of integers is passed as input to the Table Sorter class. | Note: Move back to Step 3 |
| Concluding Remarks: The *sortable* method(s) does not sort row wise and column wise in ascending order Table *t* from the null list of integers. | | | | | | |
| Testing Team: | | | Date Completed:  TBD | | | |

**$**