Black-Box Testing Assignment

**List of Requirements**

The function *public static void sortable (Table t)* should sort every row and every column of the *Table t* in ascending order.

**Supported Function**

*static Table getTable(String filename)*

returns an instance of table initialized with values from the file filename

*public int getTableValue(int row, int column)*

returns the integer at (row, column)

*public void setTableValue(int row, int column, int value)*

sets the value of the table atrow, column to value

*getSize()*

returns N

**Tests**

**Verify** *isSorted(Table t)* **Performs its Function**

1. Test *isSorted(Table t)* on random 0 x 0, 1 x 1, 2 x 2, and 3 x 3 *unsorted* matrices to make sure the method returns the correct Boolean value.
2. Test *isSorted(Table t)* on random 0 x 0, 1 x 1, 2 x 2, and 3 x 3 *sorted* matrices to make sure the method returns the correct Boolean value.

**Verify** *sortable(Table t)*

**Limiting Cases**

1. Test a null pointer exception by passing a null object as input for the *sortable* method.
2. Test an empty table by supplying the *sortable* method a table with no values contained.
3. Test max dimension of table with random values (i.e. N = Integer.MAX\_VALUE \* Integer.MAX\_VALUE).

**Boundary Cases**

1. Test a table of alphabetic characters. Should result in failure.
2. Test a table of non-alphanumeric characters (i.e. @,!#$%^. etc.). Should result in failure.
3. Test a table of float data type values, including Float.MIN\_VALUE and Float.MAX\_VALUE, using a random number generator to attain values. Should result in failure.
4. Test a table of double data type values, including Double.MIN\_VALUE and Double.MAX\_VALUE, using a random number generator to attain values. Should result in failure.
5. Test a table of long data type values, including Long.MIN\_VALUE and Long.MAX\_VALUE, using a random number generator to attain values. Should result in failure.

**Sample Unit Tests**

@Test

public static void testIsSorted(){

//create TableSorter object to access methods

TableSorter ts = new TableSorter();

//empty table test 0 x 0 matrix

int [] emptyTableVals = new int[0];

N = vals.length;

Table emptyTable = new Table(N, empyTableVals);

assertsTrue(ts.isSorted(emptyTable));

System.out.printf(“Table sorted: %s”, ts.isSorted(emptyTable)); //should return

//true

//test a null table

int [] emptyTableVals = null;

Table emptyTable = new Table(N, empyTableVals); //should throw an exception

assertsTrue(ts.isSorted(emptyTable));

System.out.printf(“Table sorted: %s”, ts.isSorted(emptyTable)); //should return

//true

//Test 1 x 1 matrix

int [] vals = { Integer.MAX\_VALUE };

N = vals.length;

Table t1 = new Table(N, vals);

assertsTrue(ts.isSorted(t1));

System.out.printf(“Table sorted: %s”, ts.isSorted(t1)); //should return

//true

//Test 1 x 1 matrix

int [] vals = { Integer.MAX\_VALUE-1 };

N = vals.length;

Table t1 = new Table(N, vals);

assertsTrue(ts.isSorted(t1));

System.out.printf(“Table sorted: %s”, ts.isSorted(t1)); //should return

//true

//Test 1 x 1 matrix

int [] vals = { Integer.MIN\_VALUE };

N = vals.length;

Table t1 = new Table(N, vals);

assertsTrue(ts.isSorted(t1));

System.out.printf(“Table sorted: %s”, ts.isSorted(t1)); //should return

//true

//Test 1 x 1 matrix

int [] vals = { Integer.MIN\_VALUE+1};

N = vals.length;

Table t1 = new Table(N, vals);

assertsTrue(ts.isSorted(t1));

System.out.printf(“Table sorted: %s”, ts.isSorted(t1)); //should return

//true

//Test 2 x 2 matrix

int [] vals = { 10, 2, //4 positives

4, 1

};

N = vals.length;

Table t1 = new Table(N, vals);

assertsFalse(ts.isSorted(t1));

System.out.printf(“Table sorted: %s”, ts.isSorted(t1)); //should return

//false

//Test 2 x 2 matrix

int [] vals = { -1, 2, //2 positives and 2 negatives

4, -11

};

N = vals.length;

Table t1 = new Table(N, vals);

assertsFalse(ts.isSorted(t1));

System.out.printf(“Table sorted: %s”, ts.isSorted(t1)); //should return

//false

//Test 2 x 2 matrix

int [] vals = { -1, 2, //3 negatives and one positive

-4, -11

};

N = vals.length;

Table t1 = new Table(N, vals);

assertsFalse(ts.isSorted(t1));

System.out.printf(“Table sorted: %s”, ts.isSorted(t1)); //should return

//false

//Test 2 x 2 matrix

int [] vals = { 6, 2, //3 positives and one negative

4, -11

};

N = vals.length;

Table t1 = new Table(N, vals);

assertsFalse(ts.isSorted(t1));

System.out.printf(“Table sorted: %s”, ts.isSorted(t1)); //should return

//false

//Test 2 x 2 matrix

int [] vals = { -1, -2, //4 negatives

-4, -11

};

N = vals.length;

Table t1 = new Table(N, vals);

assertsFalse(ts.isSorted(t1));

System.out.printf(“Table sorted: %s”, ts.isSorted(t1)); //should return

//false

//Test 2 x 2 matrix

min = Integer.MIN\_VALUE;

max = Integer.MAX\_VALUE;

int [] vals = { max, 2, //minimum and maximum integer values

4, min

};

N = vals.length;

Table t1 = new Table(N, vals);

assertsFalse(ts.isSorted(t1));

System.out.printf(“Table sorted: %s”, ts.isSorted(t1)); //should return

//false

//Test 3 x 3 matrix

int [] vals = { 9, 8, 4, //9 positives

6, 5, 1,

3, 2, 0

};

N = vals.length;

Table t1 = new Table(N, vals);

assertsFalse(ts.isSorted(t1));

System.out.printf(“Table sorted: %s”, ts.isSorted(t1)); //should return

//false

//Test 3 x 3 matrix

int [] vals = { 3, 8, -4, //4 positives and 5 negatives

-6, -5, 1,

-9, 2, -10

};

N = vals.length;

Table t1 = new Table(N, vals);

assertsFalse(ts.isSorted(t1));

System.out.printf(“Table sorted: %s”, ts.isSorted(t1)); //should return

//false

//Test 3 x 3 matrix

int [] vals = { -5, -6, -3, //9 negatives

-1, -7, -2,

-4, -8, -9

};

N = vals.length;

Table t1 = new Table(N, vals);

assertsFalse(ts.isSorted(t1));

System.out.printf(“Table sorted: %s”, ts.isSorted(t1)); //should return

//false

//Test 2 x 2 matrix

int [] vals = { 1, 2, //4 positives

3, 4

};

N = vals.length;

Table t1 = new Table(N, vals);

assertsFalse(ts.isSorted(t1));

System.out.printf(“Table sorted: %s”, ts.isSorted(t1)); //should return

//true

//Test 2 x 2 matrix

int [] vals = { -4, -3, //4 negatives

-2, -1

};

N = vals.length;

Table t1 = new Table(N, vals);

assertsFalse(ts.isSorted(t1));

System.out.printf(“Table sorted: %s”, ts.isSorted(t1)); //should return

//true

//Test 2 x 2 matrix

int [] vals = { -1, 2, //1 negative, 3 positives

3, 4

};

N = vals.length;

Table t1 = new Table(N, vals);

assertsFalse(ts.isSorted(t1));

System.out.printf(“Table sorted: %s”, ts.isSorted(t1)); //should return

//true

//Test 2 x 2 matrix

int [] vals = { -4, -2, //one positive, 3 negatives

-3, 1

};

N = vals.length;

Table t1 = new Table(N, vals);

assertsFalse(ts.isSorted(t1));

System.out.printf(“Table sorted: %s”, ts.isSorted(t1)); //should return true

//Test 2 x 2 matrix

min = Integer.MIN\_VALUE;

max = Integer.MAX\_VALUE;

int [] vals = { min, 2, //minimum and maximum integer values

4, max

};

N = vals.length;

Table t1 = new Table(N, vals);

assertsFalse(ts.isSorted(t1));

System.out.printf(“Table sorted: %s”, ts.isSorted(t1)); //should return true

//For the rest of the true 3 x 3 matrices tests populate 3 x 3 matrices with

// (positive, negative) = { (1, 8), (2, 7), (3, 6), (4, 5), (5, 4), (6, 3), (7, 2), (8, 1)}

// sets of values that are manually (known to oracle) sorted

//For example (2, 7) = 2 positive values and 7 negative values

//Test 3 x 3 matrix

int [] vals = {

-9, -8, -7,

-6, -5, -4,

-3, 2, 1

};

N = vals.length;

Table t1 = new Table(N, vals);

assertsFalse(ts.isSorted(t1));

System.out.printf(“Table sorted: %s”, ts.isSorted(t1)); //should return true

//Example 2 (4, 5) = 4 positives and 5 negatives

//Test 3 x 3 matrix

int [] vals = {

-6, -3, -1,

-5, -4, 3,

1, 2, 4

};

N = vals.length;

Table t1 = new Table(N, vals);

assertsFalse(ts.isSorted(t1));

System.out.printf(“Table sorted: %s”, ts.isSorted(t1)); //should return true

//Perform similar tests but with floats, doubles, longs

//Perform similar tests for alphabetic characters and non-alphabetic characters (i.e.

//boundary cases 1 and 2 above)

//These tests are done to establish *isTrue(Table t)* method has been tested to

//accept a “reasonable” degree of risk

}

//perform a test on empty table

@Test

public static void testSortable(){

// must import org.junit.\*; at the top of test class

// must import java.util.Arrays; at the top of test class

int [] vals = new int[0]; //create random set of values

N = vals.length; //get the amount of values, must be a square number

Table t0 = new Table (N,vals); // create a dummy table

TableSorter ts = new TableSorter(); //create a TableSorter object

ts.sortable(t0); //call the *sortable* method

dim = Math.sqrt(N);

for (int i = 0; i<dim; i++){

int[] temp = new int[dim]; //temp array to store row values

for (int j = 0; j<temp.length; j++){

temp[j] = t0.getTableValue(i,j); //retrieve row values

}

assertArrayEquals({}, t0);

}

}

//perform a test on null

@Test

public static void testSortable(){

// must import org.junit.\*; at the top of test class

// must import java.util.Arrays; at the top of test class

int [] vals = null; //create random set of values

TableSorter ts = new TableSorter(); //create a TableSorter object

assertNull(ts.sortable(t0)); //call the *sortable* method, should null exception

}

//perform a test on 1 x 1 table

@Test

public static void testSortable(){

// must import org.junit.\*; at the top of test class

// must import java.util.Arrays; at the top of test class

int max = Integer.MAX\_VALUE;

int min = Integer.MIN\_VALUE;

int [] vals\_max = {max}; //create random set of values

int [] vals\_min = {min}; //create random set of values

N\_max = vals\_max.length; //get the amount of values, must be a square number

N\_min = vals\_min.length;

Table t\_max = new Table (N\_max, vals\_max); // create a dummy table

Table t\_min = new Table (N\_min, vals\_min); // create a dummy table

TableSorter ts = new TableSorter(); //create a TableSorter object

ts.sortable(t\_max); //call the *sortable* method

ts.sortable(t\_min); //call the *sortable* method

assertTrue(ts.isSorted(t\_max));

assertTrue(ts.isSorted(t\_min));

}

//perform a test on 2 x 2 table

@Test

public static void testSortable(){

// must import org.junit.\*; at the top of test class

// must import java.util.Arrays; at the top of test class

int [] vals = {

2, 3, //create random set of values

1, 5

};

N = vals.length; //get the amount of values, must be a square number

Table t0 = new Table (N,vals); // create a dummy table

TableSorter ts = new TableSorter(); //create a TableSorter object

ts.sortable(t0); //call the *sortable* method

assertTrue(ts.isSorted(t0));

}

//perform a test on 2 x 2 table

@Test

public static void testSortable(){

// must import org.junit.\*; at the top of test class

// must import java.util.Arrays; at the top of test class

int [] vals = {

5, -3, //2 positives, 2 negatives

2, -11

};

N = vals.length; //get the amount of values, must be a square number

Table t0 = new Table (N,vals); // create a dummy table

TableSorter ts = new TableSorter(); //create a TableSorter object

ts.sortable(t0); //call the *sortable* method

assertTrue(ts.isSorted(t0));

}

//perform a test on 2 x 2 table

@Test

public static void testSortable(){

// must import org.junit.\*; at the top of test class

// must import java.util.Arrays; at the top of test class

int [] vals = {

5, 1, //3 positives, 1 negative

2, -3

};

N = vals.length; //get the amount of values, must be a square number

Table t0 = new Table (N,vals); //create a dummy table

TableSorter ts = new TableSorter(); //create a TableSorter object

ts.sortable(t0); //call the *sortable* method

assertTrue(ts.isSorted(t0));

}

//perform a test on 2 x 2 table

@Test

public static void testSortable(){

// must import org.junit.\*; at the top of test class

// must import java.util.Arrays; at the top of test class

int [] vals = {

5, -1, //1 positive, 3 negatives

-2, -3

};

N = vals.length; //get the amount of values, must be a square number

Table t0 = new Table (N,vals); // create a dummy table

TableSorter ts = new TableSorter(); //create a TableSorter object

ts.sortable(t0); //call the *sortable* method

assertTrue(ts.isSorted(t0));

}

//perform a test on 2 x 2 table

@Test

public static void testSortable(){

// must import org.junit.\*; at the top of test class

// must import java.util.Arrays; at the top of test class

int [] vals = {

-1, -2, //4 negatives

-3, -5

};

N = vals.length; //get the amount of values, must be a square number

Table t0 = new Table (N,vals); // create a dummy table

TableSorter ts = new TableSorter(); //create a TableSorter object

ts.sortable(t0); //call the *sortable* method

assertTrue(ts.isSorted(t0));

}

//Perform similar style tests for 3 x 3 matrices that were shown for the 2 x 2

//Tests should span the set of (positives, negatives) = {(1, 8), (2, 7), (3, 6), (4, 5), (5, 4), //(6, 3), (7, 2), (8, 1)}

//call sortable to sort the table of values

//int[][] temp = new int[3][3];

//manually sort the table to fill in first\_row = {…}, second\_row = {…}, and third\_row = //{…} of expected values for the assertArrayEquals() methods

//It should be noted that the *isSorted(Table t)* method is used because it has been verified //to a certain degree of risk, and more so because there are more than one way to sort a //table so long as the meet the criteria of, “*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.”*

--End of Test Plan --