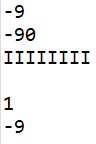
**Main and Test Output for Module 3 Programming Assignments**

Created: 06/05/21 by Tom Lever

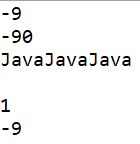
Updated: 06/05/21 by Tom Lever

**Problem 3A: Recursion in Arrays**

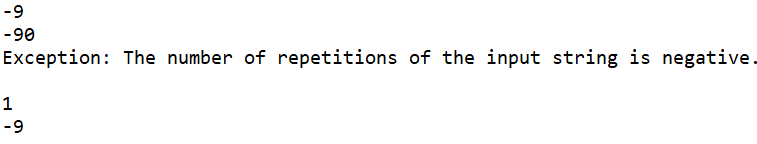
*Main Output for argument string “I 8”*



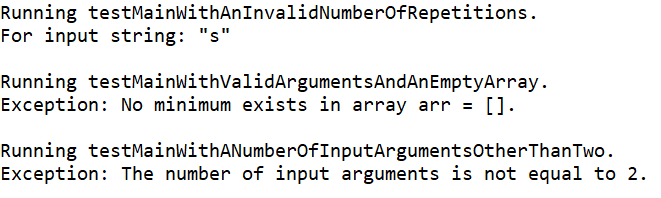
*Main Output for argument string “Java 3”*



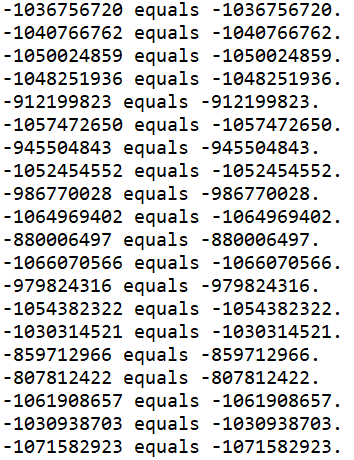
*Main Output for argument string “I -10”*



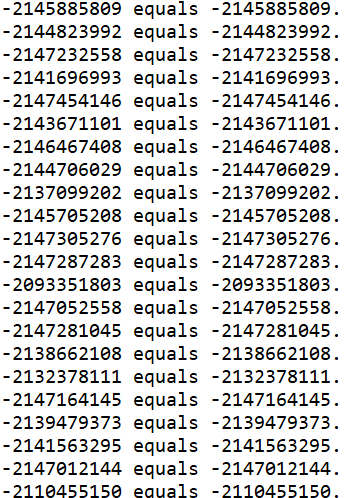
*Test Output for main*



*Test Output for smallest with One Argument*

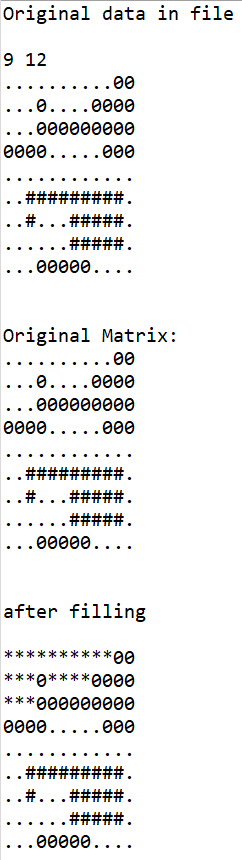


*Test Output for smallest with Two Arguments*

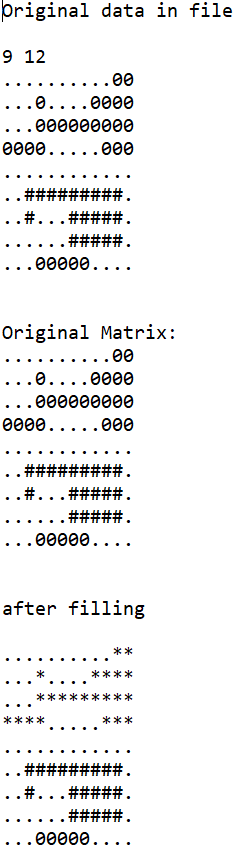


**Problem 4A: Area Fill**

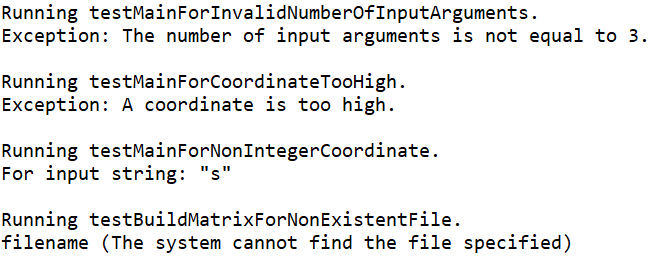
Main Output for argument string “resources/area2.txt 1 2”



Main Output for argument string “resources/area2.txt 1 3”

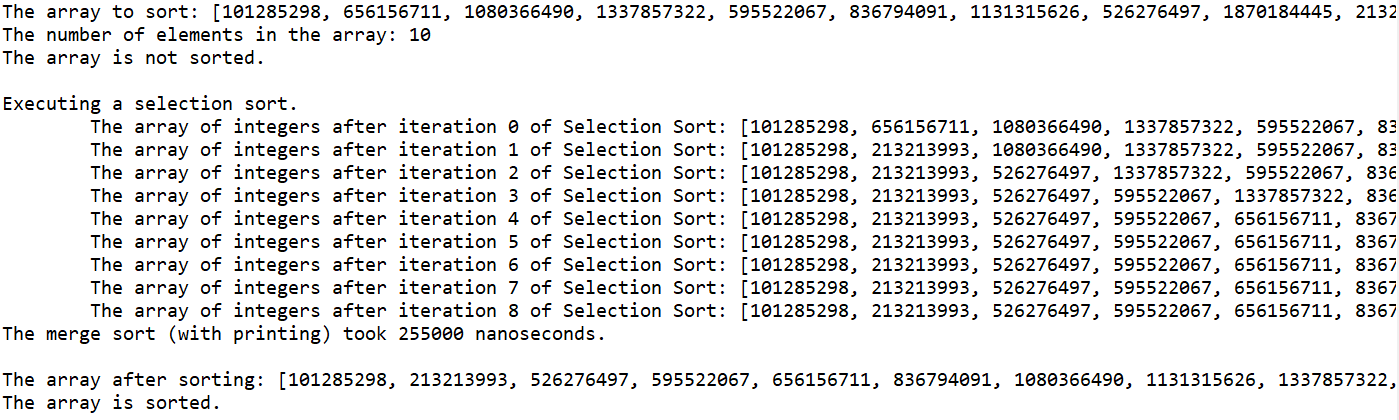


Test Output

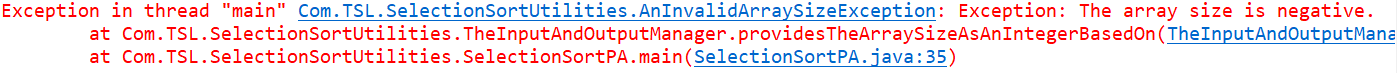


**Problem 1B: Selection Sort**

*Main Output for argument string “10”*



*Main Output for argument string “-10”*



**package** Com.TSL.SelectionSortUtilities;

**import** java.util.Arrays;

**import** org.apache.commons.math3.random.RandomDataGenerator;

/\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* SelectionSortPA encapsulates the entry point of this program, which gets an array size from a command-line argument,

\* creates an array of random integers based on the input size between zero and the maximum integer, displays the array

\* and indicates whether or not the array is sorted, executes and times a selection sort, displays the array after each

\* iteration of the sort, and displays the array and indicates whether or not the array is sorted.

\*

\* **@author** Tom Lever

\* **@version** 1.0

\* **@since** 06/06/21

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

**public** **class** SelectionSortPA

{

**private** **static** **int**[] *theArrayOfIntegers*;

/\*\* ---------------------------------------------------------------------------------------------------------------

\* main is the entry point of this program, which gets an array size from a command-line argument, creates an array

\* of random integers based on the input size between zero and the maximum integer, displays the array and

\* indicates whether or not the array is sorted, executes and times a selection sort, displays the array after each

\* iteration of the sort, and displays the array and indicates whether or not the array is sorted.

\*

\* **@param** args

\* **@throws** AnInvalidArraySizeException

-------------------------------------------------------------------------------------------------------------- \*/

**public** **static** **void** main( String[] args ) **throws** AnInvalidArraySizeException

{

**int** theArraySize = TheInputAndOutputManager.*providesTheArraySizeAsAnIntegerBasedOn*(args[0]);

*theArrayOfIntegers* = **new** **int**[theArraySize];

RandomDataGenerator theRandomDataGenerator = **new** RandomDataGenerator();

**for** (**int** i = 0; i < theArraySize; i++)

{

*theArrayOfIntegers*[i] = theRandomDataGenerator.nextInt(0, Integer.***MAX\_VALUE*** - 1);

}

System.***out***.println("The array to sort: " + Arrays.*toString*(*theArrayOfIntegers*));

System.***out***.println("The number of elements in the array: " + *theArrayOfIntegers*.length);

TheInputAndOutputManager.*printsWhetherOrNotIsSorted*(*theArrayOfIntegers*);

System.***out***.println("\nExecuting a selection sort.");

**long** theStartTime = System.*nanoTime*();

*selectionSort*(*theArrayOfIntegers*);

**long** theEndTime = System.*nanoTime*();

System.***out***.println("The merge sort (with printing) took " + (theEndTime - theStartTime) + " nanoseconds.");

System.***out***.println("\nThe array after sorting: " + Arrays.*toString*(*theArrayOfIntegers*));

TheInputAndOutputManager.*printsWhetherOrNotIsSorted*(*theArrayOfIntegers*);

}

/\*\* -------------------------------------------------------------------------------------------------------------

\* selectionSort performs a selection sort of an array of integers and displays the array after each iteration of

\* the sort.

\*

\* **@param** arr

------------------------------------------------------------------------------------------------------------- \*/

**public** **static** **void** selectionSort(**int**[] arr)

{

// Refer to the program for MergeSort in Module 3B Guided Assignment: Problem 2.

**int** theIndexOfTheLastIntegerInTheArray = arr.length - 1;

**for** (**int** i = 0; i < theIndexOfTheLastIntegerInTheArray; i++)

{

*swapTheIntegersInTheArrayAt*(

i, *getTheIndexOfTheMinimumIntegerInTheSubArrayInclusivelyBetween*(i, theIndexOfTheLastIntegerInTheArray)

);

System.***out***.println(

"\tThe array of integers after iteration " + i + " of Selection Sort: " +

Arrays.*toString*(*theArrayOfIntegers*)

);

}

}

/\*\* ---------------------------------------------------------------------------------------------------------

\* getTheIndexOfTheMinimumIntegerInTheSubArrayInclusivelyBetween gets the index of the minimum integer in the

\* subarray of the main array of integers between a first index and a second index.

\*

\* **@param** theIndexOfTheFirstIntegerInTheSubArray

\* **@param** theIndexOfTheLastIntegerInTheSubArray

\* **@return**

--------------------------------------------------------------------------------------------------------- \*/

**private** **static** **int** getTheIndexOfTheMinimumIntegerInTheSubArrayInclusivelyBetween(

**int** theIndexOfTheFirstIntegerInTheSubArray, **int** theIndexOfTheLastIntegerInTheSubArray

)

{

**int** theIndexOfTheMinimumIntegerInTheSubArray = theIndexOfTheFirstIntegerInTheSubArray;

**for** (**int** j = theIndexOfTheFirstIntegerInTheSubArray + 1; j <= theIndexOfTheLastIntegerInTheSubArray; j++)

{

**if** (*theArrayOfIntegers*[j] < *theArrayOfIntegers*[theIndexOfTheMinimumIntegerInTheSubArray])

{

theIndexOfTheMinimumIntegerInTheSubArray = j;

}

}

**return** theIndexOfTheMinimumIntegerInTheSubArray;

}

/\*\* ----------------------------------------------------------------------------------------------------

\* swapTheIntegersInTheArrayAt swaps the integers in the main array at a first index and a second index.

\*

\* **@param** theFirstIndex

\* **@param** theSecondIndex

--------------------------------------------------------------------------------------------------- \*/

**private** **static** **void** swapTheIntegersInTheArrayAt(**int** theFirstIndex, **int** theSecondIndex)

{

**int** thePlaceholder = *theArrayOfIntegers*[theFirstIndex];

*theArrayOfIntegers*[theFirstIndex] = *theArrayOfIntegers*[theSecondIndex];

*theArrayOfIntegers*[theSecondIndex] = thePlaceholder;

}

/\*\* ----------------------------------------------------

\* isSorted indicates whether or not an array is sorted.

\*

\* **@param** arr

\* **@return**

--------------------------------------------------- \*/

**public** **static** **boolean** isSorted(**int**[] theArray)

{

**for**(**int** i = 1; i < theArray.length; i++)

{

**if**(theArray[i] < theArray[i-1]) {

**return** **false**;

}

}

**return** **true**;

}

}

**package** Com.TSL.SelectionSortUtilities;

/\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* TheInputManager encapsulates functionality to provide an array size as an integer based on a command-line argument

\* and to print whether or not an array is sorted.

\*

\* **@author** Tom Lever

\* **@version** 1.0

\* **@since** 06/05/21

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

**class** TheInputAndOutputManager

{

/\*\* -----------------------------------------------------------------------------------------------------------

\* providesTheArraySizeAsAnIntegerBasedOn provides an array size based on a command-line argument. If the array

\* size parsed from the command-line argument is negative, an invalid array size exception is thrown.

\*

\* **@param** theArraySizeAsAString

\* **@return**

\* **@throws** AnInvalidArraySizeException

---------------------------------------------------------------------------------------------------------- \*/

**static** **int** providesTheArraySizeAsAnIntegerBasedOn(String theArraySizeAsAString) **throws** AnInvalidArraySizeException

{

**int** theArraySizeAsAnInteger = Integer.*parseInt*(theArraySizeAsAString);

**if** (theArraySizeAsAnInteger < 0)

{

**throw** **new** AnInvalidArraySizeException("Exception: The array size is negative.");

}

**return** theArraySizeAsAnInteger;

}

/\*\* ------------------------------------------------------------------

\* printWhetherOrNotIsSorted prints whether or not an array is sorted.

\*

\* **@param** theArray

------------------------------------------------------------------ \*/

**static** **void** printsWhetherOrNotIsSorted(**int**[] theArray)

{

System.***out***.println("The array " + ((SelectionSortPA.*isSorted*(theArray)) ? "is " : "is not ") + "sorted.");

}

}

**package** Com.TSL.SelectionSortUtilities;

/\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* AnInvalidArraySizeException represents the structure for an exception that occurs if the integer representing an array

\* size is negative.

\*

\* **@author** Tom Lever

\* **@version** 1.0

\* **@since** 05/18/21

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

**class** AnInvalidArraySizeException **extends** Exception {

/\*\* ---------------------------------------------------------------------------------------------------------------

\* AnInvalidArraySizeException() is a conventional zero-argument constructor for AnInvalidArraySizeException, which

\* calls Exception's zero-argument constructor.

--------------------------------------------------------------------------------------------------------------- \*/

AnInvalidArraySizeException() {

**super**();

}

/\*\* ---------------------------------------------------------------------------------------------------------------

\* AnInvalidArraySizeException(String message) is a one-argument constructor for AnInvalidArraySizeException, which

\* passes an error message to Exception's one-argument constructor with

\* a message argument.

\*

\* **@param** message

-------------------------------------------------------------------------------------------------------------- \*/

AnInvalidArraySizeException(String message) {

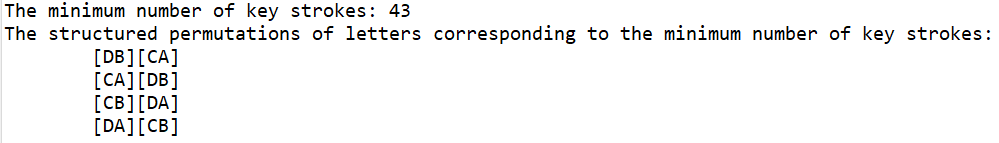
**super**(message);

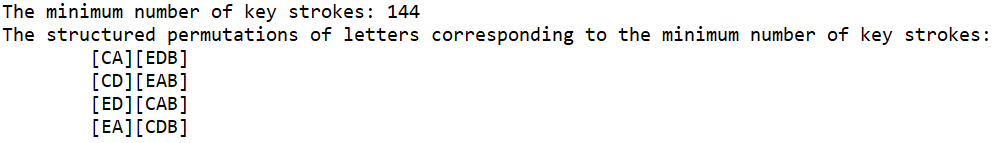
}

}

**Problem 2B: Keypad**

Main Output





Test Output

