```java
package CyrilleLingaiJonathanGrant_06;

/*
 * ComputerScience_02_01
 * Sort lists of integers by shell sorting and quick sorting algorithms.
 * JonathanGrant & CyrilleLingai
 * Integer Lists Sorting Algorithms Program_06
 * Windows 10 Eclipse IDE JRE 1.8
 *
 * Interlude: a temporary amusement or source of entertainment that contrasts with
 *             what goes before or after.
 *
 * "Computer Science is no more about computers than astronomy is about
 *  telescopes."
 * Edsger W. Dijkstra, born March 11, 1930. died August 6, 2002.
 */

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

/**
 * Read lists of integers for sorting and write the sorted lists to files.
 *
 * @author Cyrille Lingai, Grant Jonathan.
 * @version 12/05/19.
 */

public class CyrilleLingaiJonathanGrant_06 {

    // Declaring class constants.

    private final static String INPUT_FILENAME = "2050 Project 06_Input.txt";
                                    // The file of random integers.
    private final static int LIST_LENGTH = 100;
                                    // The length of the list of integers.

    /**
     * Main execution of program to scan input files, sort lists of integers,
     * and write those lists to independent files.
     *
     * @param       args                    The IO streams for reading and writing files.
     * @throws      FileNotFoundException   If the file is not found.
     * @throws      IOException             If there is not to a writable file.
     */

    public static void main(String[] args) {

        // Declaring local variables.

        int lineNumber = 0;         // Track the line an expression is scanned from.
        int fileNumber = 1;         // The file number to identify and scan files.

        int[] quickSortedArray = new int[LIST_LENGTH];
                                    // The quick sorted integers to write to file.
        int[] shellSortedArray = new int[LIST_LENGTH];
                                    // The shell sorted integers to write to file.

        String outputFileName = "2050 Project 06_OutputX.txt";
                                    // The file of corresponding sorted integers.

        File inputFile = new File(INPUT_FILENAME);  // The tool for openning files.
        FileWriter fileWriter = null;               // The tool for writing to files.
        Scanner fileScanner = null;                 // The tool to read file.

        // Get input from user and find the corresponding file.

```

```
70          try {
71
72              fileScanner = new Scanner(inputFile); // Use the reading tool on the file.
73
74              // Scan the input file for all integers and push eac integer to two lists.
75
76              while (fileScanner.hasNextInt()) {
77                  quickSortedArray[lineNumber] = fileScanner.nextInt(); // Read the line
78                  shellSortedArray[lineNumber] = quickSortedArray[lineNumber++];
79              } // End while.
80
81              fileScanner.close();
82
83          } // End try.
84          catch (FileNotFoundException e) {
85              System.err.println(e);
86          } // End catch.
87
88          // Sort both lists of integers.
89
90          shellSort(shellSortedArray);
91          quickSort(quickSortedArray, 0, LIST_LENGTH - 1);
92
93          // Attempt to create new files of the specified names,
94          // and write the sorted lists to these files.
95
96          try {
97
98              // Write the shell sorted array to file.
99
100             outputFileName = outputFileName.replace("X", Integer.toString(fileNumber));
101             fileWriter = new FileWriter(outputFileName);
102             fileWriter.write("Shell Sorted Array\n\n");
103             fileWriter.write(arrayToString(shellSortedArray));
104             fileWriter.close();
105
106             // Open a new file to write the quick sorted array.
107
108             outputFileName = outputFileName.replace(
109                     Integer.toString(fileNumber++), Integer.toString(fileNumber));
110
111             // Write the quick sorted array to file.
112
113             fileWriter = new FileWriter(outputFileName);
114             fileWriter.write("Quick Sorted Array\n\n");
115             fileWriter.write(arrayToString(quickSortedArray));;
116             fileWriter.close();
117
118         } // End try.
119         catch (IOException e) {
120             System.err.println(e.getMessage());
121         }  // End catch.
122
123     } // End main method.
124
125 // ********************************************************************************
126
127     /**
128      * Convert a sorted list of integers to a string with 10 integers per line
129      * for writing to file.
130      *
131      * @param   sortedArray The sorted list of integers.
132      * @return  arrayString The list of sorted integers for writing to file.
133      */
134
135     private static String arrayToString(int[] sortedArray) {
136
137         // Declaring local variables.
138
```

```
139         String arrayString = "";
140 
141         // Pushing each integer of the sorted list to a string for the output file.
142 
143         for (int i = 0; i < sortedArray.length;) {
144 
145             for (int j = 0; j < 10; j++, i++) {
146                 arrayString += sortedArray[i] + " ";
147             } // End for.
148 
149             arrayString += "\n";
150 
151         } // End for.
152 
153         return arrayString;
154 
155     } // End arrayToString method.
156 
157 // ****************************************************************************
158 
159     /**
160      * Sort integers using the shell sorting algorithm.
161      *
162      * @param   unsortedList    The list of unsorted integers to sort.
163      */
164 
165     private static void shellSort(int[] unsortedList) {
166 
167         // Declaring local variables.
168 
169         int nextInteger;    // The next integer in the sublist.
170         int index = 0;      // The current sublist of integers.
171 
172         // Iterate the list at intevals, dividing by 2 each time.
173 
174         for (int space = unsortedList.length/2; space > 0; space /= 2) {
175 
176             // Scan the sublist by each interval.
177 
178             for (int i = space; i < unsortedList.length; i++) {
179 
180                 // Sort the sublist.
181 
182                 nextInteger = unsortedList[i];
183 
184                 for (index = i; index >= space
185                         && unsortedList[index - space] > nextInteger;
186                         index -= space) {
187 
188                     unsortedList[index] = unsortedList[index - space];
189 
190                 } // End for.
191 
192                 unsortedList[index] = nextInteger;
193 
194             }   // End for.
195 
196         } // End for.
197 
198     } // End shellSort method.
199 
200 // ****************************************************************************
201 
202     /**
203      * Quicksort implements the textbook case of quicksort to efficiently sort the
204      * list by finding a pivot, swapping integers to the correct side of the pivot,
205      * splitting each side of the pivot into sublists, and repeating until each
206      * sublist has less than four elements. Short sublists of less than four
207      * integers are sorted using shell sort: an improved insertion sort.
```

```
208      *
209      * @param   sortingList The list or sublist of integers being recursively sorted.
210      */
211
212     private static void quickSort(int[] sortingList, int firstIndex, int lastIndex) {
213
214         // Declaring local constants.
215
216         final int MIN_SIZE = 4;         // The minimum size of the list.
217
218         // Declaring local variables.
219
220         int pivotIndex = 0;             // The pivot of the sublist.
221         int tempIndex = 0;              // Placeholder to swap two integers.
222         int midIndex = (lastIndex - firstIndex) / 2;
223                                         // Find the middle integer of the list.
224         int leftIndex = firstIndex + 1; // Leftmost integer of the sublist.
225         int rightIndex = lastIndex - 2; // Rightmostmost integer of the sublist.
226         int pivotValue = 0;             // The integer that seperates two sublists.
227
228         // Sort the sublist using insertion(shell) sort if the length is less than four,
229         // otherwise, sort the sublist by the quick sort algorithm.
230
231         if (lastIndex - firstIndex < MIN_SIZE) {
232
233             shellSort(sortingList);
234
235         } // End if.
236         else {
237
238             // Exchange integers of middle and last indices of the sublist.
239
240             tempIndex = sortingList[midIndex];
241             sortingList[midIndex] = sortingList[lastIndex - 1];
242             sortingList[lastIndex - 1] = tempIndex;
243
244             // Find the new pivot of sublist.
245
246             pivotIndex = lastIndex - 1;
247             pivotValue = sortingList[pivotIndex];
248
249             // Scan the sublist and swap integers.
250
251             while (sortingList[leftIndex] < pivotValue) {
252                 leftIndex++;
253             } // End while.
254
255             while (sortingList[rightIndex] > pivotValue) {
256                 rightIndex++;
257             } // End while.
258
259             // If the left index overlaps the right index, swap them.
260
261             if (leftIndex < rightIndex) {
262
263                 tempIndex = sortingList[leftIndex];
264                 sortingList[leftIndex] = sortingList[rightIndex];
265                 sortingList[rightIndex] = tempIndex;
266
267             } // End if.
268
269             // Swap the pivot and leftmost integer of the sublist.
270
271             tempIndex = sortingList[pivotIndex];
272             sortingList[pivotIndex] = sortingList[leftIndex];
273             sortingList[leftIndex] = tempIndex;
274             pivotIndex = leftIndex;
275
276             // Recursively quick sort the new sublists.
```

```
277
278             quickSort(sortingList, firstIndex, pivotIndex - 1);
279             quickSort(sortingList, pivotIndex + 1, lastIndex);
280
281         } // End else.
282
283     } // End quickSort method.
284
285 } // End class.
```