Text Excel Part C

Introduction 1
Getting Started 1
Tests 1
Sorting 2
Ordering TextCells 3
Ordering ValueCells, PercentCells, FormulaCells 3
Rules 3
Checkpoints 4
Checkpoint 1 4
Final 4
Hint 4
Extra Credit 4
Heterogeneous sorting – 3 points 5
Rubric 5

Introduction

In Part C we will add new commands that **sort** cells in order.

Getting Started

You must incorporate the new TextExcel C tests into your project. Start with the TextExcel_B you recently submitted. Then just grab **TestsALL.java** for Part C, and copy/paste its contents into the TestsALL.java in your project.

Tests

When you run tests, you will see two new categories: C_Checkpoint1 and C_Final. C_Checkpoint1 must pass for your checkpoint 1 submission, and both C_Checkpoint1 and C_Final must pass for your final submission. Of course, all the "A_" and "B_" tests must continue to pass for all your C submissions.

There are also some new extra credit tests. All extra credit tests are optional, and are only relevant if you are attempting the extra credit.

Sorting

You will be adding two new commands to your spreadsheet program: sort**a**, which sorts a rectangular region of cells in **a**scending order, and sort**d**, which sorts a rectangular region of cells in **d**escending order. Like all of your other commands, these commands will be interpreted in your processCommand() method. They will have the following syntax:

Command	Examples	Action
sorta <cell_range>	sorta J10-L19	Sorts the values from the range of cells in ascending order and reapplies those values into that range top to bottom, and left to right within each row, as per the example below. • The entire grid should be returned
sortd <cell_range>	sortd D6-K20	The same as sorta, except in reverse: the cells are sorted in descending order. • The entire grid should be returned

The sort commands always specify a **cell range**, which has the same meaning as the cell range included in the AVG and SUM methods of formulas. As shown above, a cell range contains two cell names, separated with a dash ('-'). The two cells identify opposite corners of a rectangle. The first cell will always be the upper-left corner, and the second cell will always be the lower-right corner.

For example, cell range `B5-E7` has been highlighted as an illustration:

```
   |A          |B          |C          |D          |E          |F
1  |3%         |           |           |           |           |
2  |           |           |hello      |           |           |
3  |           |           |2.0        |4.0        |           |3.0
4  |           |           |           |           |           |
5  |           |cars       |king       |each       |library    |
6  |           |flight     |all        |do         |joe        |
7  |           |in         |bad        |higher     |going      |
```

In the above example, after the command `sorta B5-E7` is sent, the spreadsheet then looks like this:

```
   |A          |B          |C          |D          |E          |F
1  |3%         |           |           |           |           |
2  |           |           |hello      |           |           |
3  |           |           |2.0        |4.0        |           |3.0
4  |           |           |           |           |           |
5  |           |all        |bad        |cars       |do         |
6  |           |each       |flight     |going      |higher     |
7  |           |in         |joe        |king       |library    |
```

If, instead, the command `sortd B5-E7` is sent, the spreadsheet would then look like this:

```
   |A          |B          |C          |D          |E          |F
1  |3%         |           |           |           |           |
2  |           |           |hello      |           |           |
3  |           |           |2.0        |4.0        |           |3.0
4  |           |           |           |           |           |
5  |           |library    |king       |joe        |in         |
6  |           |higher     |going      |flight     |each       |
7  |           |do         |cars       |bad        |all        |
```

You may assume each sorting command will specify a cell range containing *only* TextCells OR *only* RealCells (i.e., a mixture of ValueCell, PercentCell, and FormulaCell). You will not be asked to sort a mixture of text *and* real cells, and you will not be asked to sort a region of cells containing an EmptyCell (unless you attempt the extra credit).

Ordering TextCells

When sorting TextCells, you will order the cells in lexicographic (alphabetic) order of the String returned by their fullCellText(). You should rely on the String compareTo() method to determine this order. Bing it!

Ordering ValueCells, PercentCells, FormulaCells

When sorting RealCells, you will order the cells numerically based on what they return in their getDoubleValue method. Thus, if a user enters the following commands:

```
A20 = 30
B20 = 50%
C20 = ( 3 * 9 )
sorta A20-C20
```

then A20 is now the PercentCell of 50% (because its double value is 0.5, which is the smallest), and B20 is now the FormulaCell (because its double value is 27, the next largest), and C20 is the ValueCell of 30 (because its double value is the largest).

Note: You may assume that we will not ask you to sort FormulaCells that refer to other cells (i.e., no arithmetic formulas with cell references, and no method formulas).

Rules

You may **not** use existing sorting support such as Collections.sort() or Arrays.sort()– you must write your own sort logic. There are sorting algorithms in Chapter 13 of the book, and in the slides we went over in class. Pick your favorite algorithm and

implement it! If you are ever unsure about whether a library or method is allowed to be used for the project, ask one of the instructors.

Checkpoints

Note that the checkpoints are cumulative. When you submit each checkpoint, the submitted program should pass the details in the current checkpoint as well as all objectives in previous checkpoints. All TextExcel_A and TextExcel_B tests must continue to pass for both the checkpoint and final submission.

Checkpoint 1

- For this checkpoint, sorta must work for any rectangular region of *only TextCells*.
  - You do not have to implement sortd yet
  - You do not have to sort any other type of cell yet
  - Your TextCell need not implement the Comparable interface, but you are welcome to do so if you like.

Final

- For this checkpoint, TextCell and RealCell must implement the Comparable interface
- Your sorting algorithm should call compareTo (on the Comparable interface) to do all comparisons.
  - Your sorting algorithm should therefore *not* call any other methods on the Cells being sorted (such as fullCellText, abbreviatedCellText, getDoubleValue, etc.)
- Both sorta and sortd must work for rectangular regions of TextCells and for rectangular regions of RealCells. (We will not mix TextCells and RealCells together in a single sort command, except for the extra credit.)
- Your final submission should include the extra credit if you choose to do it.

Hint

- Your sortd algorithm will be identical to your sorta algorithm except for one tiny part. How can you use functional decomposition to reuse the entire algorithm and avoid copying / pasting code?

Extra Credit

If you choose to do the extra credit, **it is recommended that you do the rest of the project first, and that you save a version of your project without the extra credit (in case doing the extra credit makes the rest of your program not work correctly).**

Heterogeneous sorting – 3 points

An additional 3 points will be awarded for a submission which is capable of sorting rectangular regions containing any mix of cell types (EmptyCell, TextCell, ValueCell, PercentCell, FormulaCell). When there is a mix of cell types, sorta should result in all EmptyCells appearing first, then all TextCells (all ordered amongst themselves), and then finally all RealCells (ordered amongst themselves). Sortd will therefore result in the reverse (RealCells first, then TextCells, then EmptyCells).

Rubric

Requirement	Worth
TextExcel_A Functionality	2
TextExcel_B Functionality	2
Sorta algorithm	6
Sortd algorithm	6
TextCell ordering	5
RealCell ordering	5
Comparable implementation & use	5
Code Style / Comments	3
Checkpoint 1	6
Total	40
Extra credit: Heterogeneous Sorting	**3**