

Programming Project 4

Satisfied with the work you did finding SMC alumni at UC schools, your boss has come to ask for your help again.

She has a large file of students she would like to sort. It would be helpful to her to have the students sorted by the school they are attending and students attending the same school sorted by ID in ascending order. For example:

```
Joe Paarmann, 3, UCB
Otha Baloy, 5, UCB
...
Alton Hamblet, 1, UCD
Jessie Merle, 7, UCD
Lawanda Doell, 9, UCD
...
Alva Zajicek, 4, UCI
...
Chester Kemfort, 2, UCLA
Alice Mines, 6, UCLA
...
```

She tried sorting the students herself but when she sorts by school the students are no longer sorted by ID within each school, and vice-versa. Having learned of stable sorting you agree to take on this simple task.

Part 1:

You plan to use the Java library stable sort to do the sorting. Since the sort is stable, you can do two consecutive sorts and end up with the needed result (note that which sort you do first matters).

To sort a list by some criteria in Java 7, you use the `Collections.sort()` method like below. The second argument is an implementation of the `Comparator` interface, which determines how two elements are compared.

```
Collections.sort(myList, myComparator);
```

Part 2:

It occurs to you that you don't need two separate sorts. You plan to write a single comparator that sorts by school and breaks ties by ID and use that to sort only once to get the desired result.

Part 3:

Both parts 1 and 2 were simple and fast enough, but you have a hunch you could probably make this a tiny bit faster. You decide to implement your own simple algorithm for the problem based on bucket sort. The idea is very simple:

- You create one bucket (a list) for each school.
- You do one pass through the input list and place each student you encounter into its corresponding bucket.
- You sort each of the buckets (by which field?) by using a comparator like in Part 1.
- You go through each bucket in order and put the students one by one into the original list.

Hints:

A lot of the code has been given. **All you need to do is fill out the empty bodied methods.**

Make sure you are comfortable using the ArrayList class (we have seen most methods you will use in the lists lecture) and doing a simple sort using a comparator. You may want to try that with a list of strings first, just for some quick practice. API docs are here:

<http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html>

<http://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html> // do not implement equals() method

For Part 3 make sure you really understand the algorithm (simple as it is) before you get down to writing any code. Do not write even one line of code if you are not very clear what needs to be done (best achieved by doing a simple example on paper – no code involved at all, just a good diagram).

Deliverables:

You should submit a zip file named project4_first_last.zip (where first and last are your first and last name) containing **ONLY** the 2 files below.

Project4.java

report.txt - a text (not Word, etc.) file containing:

- 1) a 1 to 10 lines paragraph from you saying “I have tested this program and there are no known issues.” if you believe that to be the case, or a brief description of known issues in case your program has known problems or you could not fully implement it.
- 2) a copy of the program output (from the main function) containing the **timing printouts** and the **first 3 and last 3 lines** from **your bucket sort implementation's** studentsSorted.txt file.
- 3) a **short and to the point** (6-10 lines max) **runtime** and **space** analysis for your bucket sort (you can assume each of the k buckets is the same size and that the library sort is merge sort). Give **big O for space and time**. Would the runtime be better or worse if k were larger than 7?

How you get points:

sortBySchoolById1	15 points
sortBySchoolById2	20 points
sortBySchoolById3	50 points
sortBySchoolById3 analysis	15 points

How you lose points:

- You do not follow the given directions and decide to make changes “for fun”. Specifically, **do not change the code given to you**. Can use helper methods if you need to (you shouldn't really need to) but do not change the given code.
- You keep separate bucket variables for each school in Part 3. That is probably doable if you have 7 schools but would not be if we had 100 schools for example. **You must use an array for the buckets. Your algorithm must be very easily modifiable to support k schools.**
- Your algorithm for Part 3 is inefficient.
- You submit your whole workspace. **Submit only the files the project asks for.**
- If any of your code prints anything at all on the console except for the timing output. **Remove all your print outs, debug statements, etc. Clean up your code and do not leave clutter behind.**
- Your code has no comments where needed. **Comment your code appropriately.**