Homework 07 – Merge-Mania and More

Authors: Brian, Harry, Faris, Chelsea
Topics: Recursion and Merge-sort

Problem Description

Please make sure to read the document fully before starting!

In this assignment, you will be tasked with writing methods that will solve some puzzles recursively. For some of them, you will need to use the merge method from *RecursionUtils.java*, which will be provided. **NOTE:** you can ignore any Checkstyle errors in *RecursionUtils.java* and *Point.java* since they are provided code files.

Solution Description

For this assignment, you will create the following class: *Recursion.java*

Notes:

1. *You will be able to use provided helper methods from RecursionUtils.java and Point.java. Review the Javadoc for those methods.*
2. *The methods in this assignment should be public and static, as they are procedures that do not use any instance data.*
3. *All methods in this assignment must use recursion or call recursive helper methods.*
    a. **INDICATE WHICH METHODS ARE RECURSIVE IN YOUR COMMENTS**
4. *Make sure to add all Javadoc comments to your methods*

Recursion.java

This class will hold all the recursive methods you will be implementing.

Methods:

- ~~`mergeSort`~~
    - ~~Takes in a String[] and returns a String[] containing the elements of the input sorted in ascending lexicographical order.~~
    - ~~The input array will not be null or contain null entries.~~
    - ~~This algorithm must sort the input array using merge sort, as taught in lecture.~~
    - ~~When merging two arrays, you **must** use the `merge` method from *RecursionUtils.java*.~~
    - ~~Example 1:~~
        - ~~Input: [“Brian”, “Alex”, “David”, “Charlie”, “Aaron”]~~
        - ~~Output: [“Aaron”, “Alex”, “Brian”, “Charlie”, “David”]~~
    - ~~**HINT:** You may the `copyOfRange` method helpful in *RecursionUtils.java*~~
- `mergeAll`
    - Takes in a String[][], where each index stores an ascending sorted array of Strings.

    - The input array will not be *null* or contain *null* entries.
    - Returns a String[] containing all Strings from each index of the input in ascending lexicographical order.
    - When merging two arrays, you **must** use the `merge` method from *RecursionUtils.java*.
    - This method must be recursive or call recursive helper methods.
    - Example:
        - Input: [[“Alex”, “Brian”], [“Aaron”, “David”], [“Charlie”]]
        - Output: [“Aaron”, “Alex”, “Brian”, “Charlie”, “David”]
- ~~`countDuplicates`~~
    - ~~Takes in a String[], which will be sorted in ascending lexicographical order.~~
    - ~~Returns an int, representing the number of duplicate elements in the input array.~~
    - ~~This method **must** be recursive, meaning that it will call itself at some point.~~
    - ~~Example:~~
        - ~~Input: [“A”, “A”, “B”, “C”, “C”, “C”, “D”]~~
        - ~~Output: 3~~
        - ~~Reason: “A” is duplicated once, and “C” is duplicated twice. All other elements are unique. Thus, there are a total of three duplicates in the input.~~
- `verifyPalindrome`
    - Takes in a String.
    - Returns a boolean, representing if the input string was a palindrome.
    - If the input is an empty string, return true.
    - If the input is null, return false.
    - A palindrome is a String that reads the same forwards and backwards
        - Ex: The word “racecar” read backwards is still “racecar”. Therefore, racecar is a palindrome.
    - This method **must** be recursive, meaning it will call itself at some point
    - This method should check if the String is palindrome **regardless** of casing (i.e. it should be case-insensitive).
    - Example:
        - Input: Civic
        - Output: true, “Civic” backwards is “civiC”
    - Example:
        - Input: “java”
        - Output: false, “java” backwards is “avaj”
    - Example:
        - Input: “tacocat”
        - Output: true, “tacocat” backwards is “tacocat”
- `numInteriorPoints`
    - Takes in a Point[] and an int.
    - The first parameter (Point[]) represents the Cartesian coordinates for a set of points.
        - Each index stores a `Point`, representing the coordinate pair of a point.
        - The `x` field represents the x-coordinate.
        - The `y` field represents the y-coordinate.
    - The second parameter (int) represents the radius of a circle centered at the **origin**.
    - Return an int representing the number of points that are **strictly within the circle**.
        - **HINT:** Think about how you can use the Pythagorean Theorem in this situation

- ~~This method **must** be recursive or use recursive helper methods.~~
- ~~Example:~~
    - ~~Input: [(1, 1) (0, 0) (3, 4) (5, 5)], 5~~
    - ~~Output: 2, the first two points are within the circle of radius 5. The last two points have distance of 5 or more from the origin, and thus are not inside of the circle~~

~~Checkstyle~~

You must run Checkstyle on your submission (To learn more about Checkstyle, check out cs1331-style-guide.pdf under CheckStyle Resources in the Modules section of Canvas.) **The Checkstyle cap for this assignment is 40 points.** This means there is a maximum point deduction of 40. If you don't have Checkstyle yet, download it from Canvas -> Modules -> CheckStyle Resources -> checkstyle-8.28.jar. Place it in the same folder as the files you want to run Checkstyle on. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-8.28.jar yourFileName.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off (limited by the checkstyle cap mentioned in the Rubric section). Note that you can ignore any Checkstyle errors in code we provide you, where applicable. In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

Additionally, you must Javadoc your code.

Run the following to only check your Javadocs:

```
$ java -jar checkstyle-8.28.jar -j yourFileName.java
```

Run the following to check both Javadocs and Checkstyle:

```
$ java -jar checkstyle-8.28.jar -a yourFileName.java
```

For additional help with Checkstyle see the CS 1331 Style Guide.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `Recursion.java`

Make sure you see the message stating the assignment was submitted successfully. From this point, Gradescope will run a basic autograder on your submission as discussed in the next section. **Any autograder test are provided as a courtesy to help "sanity check" your work and you may not see all the test cases used to grade your work.** You are responsible for thoroughly testing your submission on your own to ensure you have fulfilled the requirements of this assignment. If you have questions about the requirements given, reach out to a TA or Professor via Piazza for clarification.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your latest submission. **Be sure to submit every file each time you resubmit**.

Gradescope Autograder

If an autograder is enabled for this assignment, you may be able to see the results of a few basic test cases on your code. Typically, tests will correspond to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g. non-compiling code)
- Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or Checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Allowed Imports

To prevent trivialization of the assignment, you are not allowed to use any imports.

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`
- `System.arraycopy`

Collaboration

Only discussion of the Homework (HW) at a conceptual high level is allowed. You can discuss course concepts and HW assignments broadly, that is, at a conceptual level to increase your understanding. If you find yourself dropping to a level where specific Java code is being discussed, that is going too far. Those discussions should be reserved for the instructor and TAs. To be clear, you should never exchange code related to an assignment with anyone other than the instructor and TAs.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- **Check on Ed Discussion for a note containing all official clarifications and sample outputs**

It is expected that everyone will follow the Student-Faculty Expectations document, and the Student Code of Conduct. The professor expects a **positive, respectful, and engaged academic environment** inside the classroom, outside the classroom, in all electronic communications, on all file submissions, and on any document submitted throughout the duration of the course. No inappropriate language is to be used, and any assignment, deemed by the professor, to contain inappropriate, offensive language or threats will get a zero. You are to use professionalism in your work. Violations of this conduct policy will be turned over to the Office of Student Integrity for misconduct.