Programming Exercise 05 – Merge-Mania and More

Topics: recursion, merge-sort

Problem Description

Please read this document in its entirety before writing any code!

Recursion is the process of solving a problem by solving smaller versions of that problem. In other words, the solution to the original problem depends on the solutions to smaller problems, which depend on the solutions to even smaller problems. This process continues until we reach a problem that can no longer be broken down into a smaller problem, called the base case.

In this assignment, you will be tasked with solving some puzzles recursively by writing recursive methods. For some methods, you will need to use the `merge` method from *RecursionUtils.java*, or the methods from *Point.java*, both of which will be provided. Although some of these methods may seem much more intuitive and straight-forward to implement iteratively, the purpose of this assignment is to expose you to the process of solving problems recursively so that you are better prepared to encounter applications that are more naturally recursive.

Recall that a recursive method involves three components:

- **Base case:** What is the smallest version of this problem? The solution to the base case is typically trivial, and there may be multiple base cases.
- **Approach the base case:** How can the problem be broken down into smaller problems? This often involves doing some "work" to make the problem a smaller one and may also involve passing in new or different parameters to the recursive call.
- **Recursive call:** The method makes a call to itself, which is what makes the method recursive!

Solution Description

For this assignment, you will create one file: *Recursion.java*, and implement several static methods. For each method, we have listed requirements along with some hints towards a suggested approach. We encourage you to follow these hints if you need guidance, but you are also welcome to come up with your own recursive solutions.

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 be recursive or call private recursive helper methods.**

Recursion.java

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

Methods:

- `mergeSort`
  - Takes in a `String[]` and returns an ascendingly sorted `String[]` containing all the Strings in the input array.
    - You may assume that the input array will not be null nor contain null elements.
  - Your implementation must be merge sort as taught in lecture and run in $O(n \log n)$.
  - When merging two arrays, you **must** use the `merge` method from *RecursionUtils.java*.
  - **Example**
    - **Input:** `[“Brian”, “Alice”, “David”, “Chloe”, “Aaron”]`
    - **Output:** `[“Aaron”, “Alice”, “Brian”, “Chloe”, “David”]`
  - **Hints**
    - **Base cases:** The most trivial array to merge sort is an array of length 0 or 1 because these arrays cannot be halved.
    - **Approach the base case:** By halving the input array with each recursive call, we will eventually reach the base case. The `copyOfRange` method will be helpful!
    - **Recursive call:** Call `mergeSort` twice to sort each half.
    - After the recursive call, you should have two sorted halves that can be passed into the `merge` method to produce a merged sorted array.
- `mergeAll`
  - Takes in a `String[][]`, where each index refers to an ascendingly sorted `String[]`.
    - You may assume that the input array will not be null nor contain null elements, and that the array at each index will also not contain null elements.
  - Returns an ascendingly sorted `String[]` containing all the Strings from the array at each index of the input.
  - When merging two arrays, you **must** use the `merge` method from *RecursionUtils.java*.
  - **Example**
    - Input: `[ [“Alice”, “Brian”], [“Aaron”, “David”], [“Chloe”] ]`
    - Output: `[“Aaron”, “Alice”, “Brian”, “Chloe”, “David”]`
  - **Hints**
    - It is recommended to write a private helper method that does the heavy lifting for this method. Here is the suggested return type and method signature:

      ```
      String[] rMergeAll(String[][], int)
      ```

      - The `String[][]` parameter represents the array of arrays to merge.
      - The `int` parameter represents the starting index of the 2-D array being considered, with the first call starting at 0.
    - **Base Case:** When the index is beyond the length of the 2-D array, there are no more arrays to merge and the “merged” array would be an empty array.
    - **Approach the Base Case:** By incrementing the index that is passed in with each recursive call, there will eventually be no more arrays to merge.
    - **Recursive Call:** Call `rMergeAll` and pass in the incremented index. This represents the process of merging the rest of the arrays.
    - After the recursive call, merge the array at the index with rest of the arrays.

- `countDuplicates`
  - Takes in a `String[]` which will be sorted in lexicographically ascending order.
  - Returns an `int`, representing the number of duplicate elements in the input array.
  - **Example**
    - Input: `[“A”, “A”, “B”, “C”, “C”, “C”, “D”]`
    - Output: 3
      - “A” is duplicated once, and “C” is duplicated twice.
  - **Hints**
    - **Base case:** The simplest array to count duplicates of is an array of length 0 or 1.
    - **Approach the base case:** By making a copy of the array without the first String in each recursive call, there will eventually be less than two Strings to compare.
    - **Recursive call**: Call `countDuplicates` and pass in the subarray. The `copyOfRange` method will be helpful!
    - After the recursive call, count the number of duplicates between the first two Strings, and add that to the number of duplicates in the rest of the array.
- `verifyPalindrome`
  - A palindrome is a string that reads the same forwards as it does backwards.
    - E.g., “racecar” backwards is still “racecar”, so racecar is a palindrome.
  - Takes in a `String` and returns a `boolean` representing if the input string is a palindrome.
    - If the input is null, return false.
    - If the input is an empty string, return true.
  - This method should check if the string is a palindrome **regardless** of casing (i.e., it should be case-insensitive).
  - **Examples**
    - Input: “Civic”
    - Output: true, “Civic” backwards is “civiC”

    - Input: “java”
    - Output: false, “java” backwards is “avaj”

    - Input: “tacocat”
    - Output: true, “tacocat” backwards is “tacocat”
  - **Hints**
    - **Base cases:** A String of length 0 or 1 can easily be verified to be a palindrome.
    - **Approach the base case:** By creating a substring that does not include the first and last characters of the string with each recursive call, there will eventually be less than two characters to compare.
    - **Recursive call**: Call `verifyPalindrome` and pass in the substring.
    - Compare the first and last characters of the input string, and combine the result of that comparison with whether the substring is a palindrome.

- `binarySearch`
  - Takes in a `String[]` containing unique elements sorted in lexicographical ascending order representing the array to search in, and a `String` representing the target String to search for.
    - You may assume the input array will not be null nor contain null elements.
  - Returns an `int` representing the index of the target String in the input array.
    - If the target String is not found, return -1.
  - Your implementation must be binary search as taught in lecture and run in $O(log\ n)$.
  - **Examples**
    - Input: `[“Aaron”, “Alice”, “Brian”, “Chloe”, “David”], “Chloe”`
    - Output: 3
    - Input: `[“Aaron”, “Alice”, “Brian”, “Chloe”, “David”], “Emily”`
    - Output: -1
  - **Hints**
    - It is recommended to write a private helper method that does the heavy lifting for this method. Here is the suggested return type and method signature:

      ```
      int rBinarySearch(String[], String, int, int)
      ```

      - The `String[]` parameter represents the array to search in.
      - The `String` parameter represents the String to search for.
      - The two `int` parameters represents the starting and ending indices of the range of the array to consider, both inclusive.
    - **Base case:** When the start index is greater than the end index, the element was not found and there are no more elements to search for.
    - **Approach the base case:** By modifying the start or end index of the array to consider with each recursive call, we will eventually reach the base case.
    - **Recursive call:** Call `rBinarySearch` and pass in the appropriate parameters depending on the result of the comparison.

Checkstyle

You must run Checkstyle on your submission (To learn more about Checkstyle, check out cs1331-style-guide.pdf under the Checkstyle Resources module on Canvas). **The Checkstyle cap for this assignment is 50 points.** This means there is a maximum point deduction of 50. 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). In future assignments 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 the class forum 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 assignment. 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 may not import any classes for this assignment.

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 assignment at a conceptual high level is allowed. You can discuss course concepts and 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.

You **MAY NOT** use code generation tools to complete this assignment. This includes generative AI tools like ChatGPT.

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.