CMPSCI 187 / Spring 2015
Sorting Kata

Due on Thursday, April 30, 8:30 a.m

Marc Liberatore and John Ridgway

Morrill I N375
Section 01 @ 10:00
Section 02 @ 08:30

Contents

Overview

In this assignment, you will implement three in-place sorting algorithms: insertion sort, heap sort, and quick sort. Your implementations will operate on a list with the minimal set of methods necessary to support a comparison-based sort (`compare`, `swap`, and `size`), and will not be able to directly examine the values in the list.

Learning Goals

- Demonstrate understanding of sorting algorithms by implementing them.
- Show ability to work with a restricted list abstraction.
- Finally be done with the relentless torrent of CMPSCI 187 assignments.

General Information

Read this entire document. If, after a careful reading, something seems ambiguous or unclear to you, then email cs187help@cs.umass.edu immediately.

Start this assignment as soon as possible. Do not wait until 5pm the night before the assignment is due to tell us you don't understand something, as our ability to help you will be minimal.

Reminder: Copying partial or whole solutions, obtained from other students or elsewhere, is academic dishonesty. Do not share your code with your classmates, and do not use your classmates' code.

You are responsible for submitting project assignments that compile and are configured correctly. If your project submission does not follow these policies exactly you may receive a grade of zero for this assignment.

Policies

- For some assignments, it will useful for you to write additional class files. Any class file you write that is used by your solution MUST be in the provided `src` directory you export.
- The TAs and instructors are here to help you figure out errors, but we won't do so for you after you submit your solution. When you submit your solution, **be sure to remove all compilation errors from your project**. Any compilation errors in your project will cause the autograder to fail, and you will receive a zero for your submission.

Test Files

In the `test` directory, we provide several JUnit test cases that will help you keep on track while completing the assignment. We recommend you run the tests often and use them to help create a checklist of things to do next. But you should be aware that we deliberately don't provide you the full test suite we use when grading.

We recommend that you think about possible cases and add new `@Test` cases to these files as part of your programming discipline. Simple tests to add will consider questions such as:

- Do your methods taking integers as arguments handle positives, negatives, and zeroes (when those values are valid as input)?
- Does your code handle unusual cases, such as empty or maximally-sized data structures?

More complex tests will be assignment-specific. To build good test cases, think about ways to exercise methods. Work out the correct result for a call of a method with a given set of parameters by hand, then add it as a test case. Note that we will not be looking at your test cases, they are just for your use.

Before submitting, make sure that your program compiles with and passes all of the original tests. If you have errors in these files, it means the structure of the files found in the `src` directory have been altered in a way that will cause your submission to lose some (or all) points.

Problem 1

Import Project into Eclipse

Begin by downloading the starter project and importing it into your workspace. It is very important that you **do not rename** this project as its name is used during the autograding process. If the project is renamed, your assignment will not be graded, and you will receive a zero.

The imported project may have some errors, but these should not prevent you from getting started. Specifically, we may provide JUnit tests for classes that do not yet exist in your code. You can still run the other JUnit tests.

The project should normally contain the following root items:

`src` This is the source folder where all code you are submitting must go. You can change anything you want in this folder (unless otherwise specified in the problem description and in the code we provide), you can add new files, etc.

`support` This folder contains support code that we encourage you to use (and must be used to pass certain tests). You must not change or add anything in this folder. To help ensure that, we suggest that you set the support folder to be read-only. You can do this by right-clicking on it in the package explorer, choosing Properties from the menu, choosing Resource from the list on the left of the pop-up Properties window, unchecking the Permissions check-box for Owner-Write, and clicking the OK button. A dialog box will show with the title "Confirm recursive changes", and you should click on the "Yes" button.

`test` The test folder where all of the public unit tests are available.

`JUnit 4` A library that is used to run the test programs.

`JRE System Library` This is what allows Java to run; it is the location of the Java System Libraries.

If you are missing any of the above or if errors are present in the project (other than as specifically described below), seek help immediately so you can get started on the project right away.

Problem 1 continued on next page...

Examining the code

Ultimately, you will be implementing the `sort` method in each of three classes: `InsertionSorter`, `HeapSorter`, and `QuickSorter`. But before you do so, spend a few minutes to look over the code we've provided.

In the `comparators` package, we provide two comparators that do what you'd expect on `Integer`s and `String`s. These are used in the tests we provide.

In the `structures` package, we provide an interface (and accompanying implementation) of a `SwapList`. Your sorting algorithm implementations will have to work within this interface. Notably, you cannot directly examine values in the underlying list; you can only `compare` and `swap` them. We do provide a `toString` method, but **do not use its results in your sorting implementation** — it is for debugging only. We will treat attempts to parse and sort the returned `String` as an attempt to cheat, and at best you'll receive a zero for this assignment.

In the `sorters` package, look first at the `AbstractSorter` class. Each of the sorters you will implement should extend this class. They will have access to its **`protected`** elements: the `list` to sort and the `comparator` to be used when sorting.

We've provided a complete example of a concrete implementation of a sorter in `SelectionSorter`. It is a close adaptation of the code for selection sort that we covered in lecture (and that DJW provides). Your sorter implementations will look something like this, manipulating the `list` and likely using one or more **`private`** methods.

What to do

Implement the `sort` method in each of `InsertionSorter`, `HeapSorter`, and `QuickSorter`, using the corresponding algorithm. For the last, note that we've specified which index's value you must use as the pivot.

If your implementations differ from the ones presented in lecture, you may find the provided tests that check the number of swaps and comparisons may be off slightly. Being off slightly is OK — the graded tests will allow for some leeway. But if they're off significantly and you don't know why, it may indicate you've implemented the algorithm incorrectly. Don't micro-optimize them; just implement the algorithms from lecture or the book as-is.

If we find that you've implemented the wrong algorithm (for example, you copy/paste the code from `SelectionSorter.sort` into each of the other classes), we will assume you're attempting to game the autograder. **We will treat this as an attempt to cheat,** and at best you'll receive a zero for this assignment.

Export and Submit

When you have completed the changes to your code, you should export an archive file containing the entire Java project. To do this, click on the `sorting-kata-student` project in the package explorer. Then choose "File → Export" from the menu. In the window that appears, under "General" choose "Archive File". Then choose "Next" and enter a destination for the output file. Be sure that the project is named **`sorting-kata-student`**. Save the exported file with the `zip` extension (any name is fine). Log into Moodle and submit the exported zip file.