Assignment 5: The Variety Pack

Pre-Assignment Notes

The **late penalty** is -10% per day late. After two days, late submissions won't be accepted.

This is an **individual assignment**, meaning you're not supposed to share or copy your solution/code from other students or the internet. There's still the class's Piazza if you want to ask questions to the group, or you can ask during lecture, send an email to the class's help account, or attend an office hour on Discord.

Note you can use any code shared by the instructor or anything from class or the tutorial videos linked on the course website. You *can* get outside help from guides and other people, just try to make sure they're teaching you how to solve the assignment, not writing the code for you. You won't learn anything that way, and they can't write your exam or midterm for you. If in doubt about whether something is plagiarism or not, don't be afraid to reach out and ask!

1. ...And The Rest!

We're near the end of the course now, and we'll be ranging between a few different topics in the last few weeks from a number of different areas. As such, instead of picking one of those subjects to make the centrepiece of our last assignment, you'll be solving two smaller problems drawn from different sub-sections of the course material.

First off, there's our return to Trees, which both introduced new varieties and expanded on previously introduced ones. While their structures and internal implementations vary, this forest of Trees are all trying to offer you O(log n) search, insertion, and removal times for key-bearing entries, based on how they control their height. As such, they're good candidates for implementing other key-based data structures (particularly Maps), but with limited support in standard Java, you'd have to do most of the implementation work yourself.

Secondly, the sorting algorithms. Some of the most important and fundamental algorithms in computer science, many of the structures we've considered over the course wouldn't be possible (or would be much less useful) without a means of sorting data. The different algorithms have different approaches and advantages, however, and

Don't worry if anything this assignment's asking you to do doesn't sound familiar yet – you might just be ahead of the game! We'll get there.

2. Assignment Outline

Just two questions this time. Each problem is independent of the other, although including them in the same IntelliJ project and creating one consolidated main method for demonstrating your work is still a good idea!

2.1 Searching for the Middle Ground

A binary search tree is a solid data structure for finding entries based on their key, even if it isn't an AVL and enforcing balance to keep the search time under control. Without knowing what key you're looking for, however, the data structure can be a little less intuitive to navigate.

You're going to build and implement your own BST class, using integers sorted smallest to largest as the keys. This class should include a Tree Search function that can find an entry given a key, an insertion function that can take in a new integer as a key and add a new entry where it belongs, and a removal function that can remove a given entry.

You're also going to add a function that prints the mean and median of the stored keys in the binary search tree. This function should certainly run in no worse than O(n), and should be possible in about n/2 (times some constant for the primitive operations, of course). So for example, a three-node Tree of the keys 4, 6, and 9, would print a mean of 6.33 and a median of 6.

Finally, include a function that can print the contents of the tree to the terminal. You may choose to represent this however you wish, such as giving each node its own line where you print its key, parent, and children, to some sort of visualization using tabs or arrows, so long as it's clearly legible about the nodes of the tree and how they're connected.

You should also include a main method that builds a tree one node at a time, printing the tree after each node added along with the current mean and median for the stored keys. The main method should also allow the user to add a node by submitting a new key and seeing an updated tree, mean, and median printed.

While you should feel free to use existing data structures or functions to help you, note that any data structure that lets you do a complete end-run around actually building a BST (for example, building a BST-wrapper around an ArrayList) will lose your solution some structure marks!

2.2 Sorting Itself Out

Choosing the best sorting algorithm for a job can sometimes be tricky without trying a few out first to see what they look like, or even running some experiments to collect some data to see if your reasoning holds in practice.

To that end, you're going to implement two different functions for sorting a 32-bit sequence of zeroes and ones, such that all of the zeroes are sorted in front of all of the ones. For example, an input sequence of 00101101010001010100101010101011 should come out as 00000000000000000111111111111111. Exactly how to represent and handle this sequence is up to you, although an array of ints is a perfectly valid and straightforward choice.

One sorting function should be based on merge-sort, while the other will be based on quick-sort. Before implementing either function, write up which one you expected to be more efficient, and why. Your answer should be between a paragraph and a half-page, and leverage what you know about each algorithm.

Once implemented, both functions should also track how many comparisons it takes before they can certify that a sequence is sorted, which you should use as your performance metric for time taken. Write an accompanying main method that tests 100 randomly-generated sequences sorted by your methods, printing how many comparisons each function took on average to sort the sequence. Finally, take these experimental results and write a follow-up to your prediction and whether it bore out. This follow-up should also be between one paragraph and half a page.

3. Deliverables

All assignment submissions will be done through CourSys at https://courses.cs.sfu.ca/. For this assignment, that should include:

1. A .zip archive of your code, organized into a single Java project. To ensure compatibility, it's recommended to use IntelliJ while developing your code, then using the export function under File -> Export -> Project To .zip File. **Remember to include the necessary test methods!**

2. A .pdf of your written analysis for part 2.2. Please submit this as a single document in the separate available field on CourSys, rather than bundled within the code .zip file.

Be sure you tidy up your code before submitting! Check the Java code style guide posted on the course website, and do your best to provide helpful comments.