A2-Structured Insertion Sort

Stuart Thiel

May 22, 2023

Introduction

This assignment is worth **5%** of your grade. This is an **individual assignment** (so individual, I put *your* student ID on it) and you should not share your assignment with anyone else. The **assignment is due June 2nd at 11:59pm, Montreal time**.

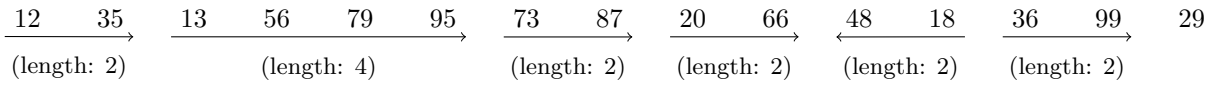
All submissions must go through Moodle. I like EAS, but the new VPN rules suck.

Q1) Structured Insertion Sort

Your program, called `StructSort.java`, should define a single class, StructSort in the `comp352.assignment.two` package. Its main method will take one argument, the name of a file containing the input data. The input data should consist of positive integers separated by spaces. The program itself will perform a "structuring pass" on the input data before doing a regular insertion sort. Once complete, your program should output various statistics (specified below) and the sorted results.

The purpose of the "structuring pass" is to improve the efficiency of the regular insertion sort. During the "structuring pass," the program identifies sequences of numbers that are either increasing or decreasing in order. The program finds the longest possible "run" in the specified direction in a "greedy" fashion, meaning it moves on to the next "run" only after it has identified the longest possible "run" in the current direction. If a set of consecutive values are in *ASCENDING* order, then the program must reverse the order of those values using an algorithm that changes the order of the values to the opposite direction. For example, if a *ASCENDING* run `"2, 3, 7, 10"` were found, your program should reverse it to the *DESCENDING* run `"10, 7, 3, 2"`.

Your program should keep track of how many times you reverse values during the "structuring pass" and also keep track of any runs in *DESCENDING* order of length 2. For example:



If the "greedy" run detection leaves a run of length one at the end, it is neither ascending nor descending and should not be counted.

The final output should be sorted in *DESCENDING* order.

DebugRunner is a tool that provides feedback on the program. It can be run from the command line using "java DebugRunner" to check if the program meets the requirements of the assignment. To do this test, it examines a `config.xml` file to interpret how to call your program and evaluate the results.

1.1 Guidance for Structured Insertion Sort

Understanding Structured Insertion Sort: Begin by familiarizing yourself with the standard insertion sort algorithm. This assignment requires you to add a structuring pass to the standard insertion sort. This structuring pass includes identifying sequences or "runs" of numbers that are either increasing or decreasing and reversing them based on certain criteria.

Implementing the Structuring Pass: Identify the longest possible run in a "greedy" fashion, i.e., only after identifying the longest possible run in the current direction, move on to the next run. You need to implement this in your code.

Reversing the Runs: When a run is found in a specified order, you will have to reverse its order. Be sure to understand how to correctly identify and reverse these runs in your code.

Outputting Relevant Statistics: Make sure your program keeps track of and outputs the required statistics like the number of times you reversed the order of values during the structuring pass, or any runs in a specified order of a certain length.

Debugging and Testing: Use the provided DebugRunner tool and the config.xml file to debug and test your code. The DebugRunner tool provides feedback on whether your program meets the requirements of the assignment.

Reflect on the Impact: Consider the impact of the structuring pass, specifically the chosen reversals on the number of swaps and comparisons, as well as the impact of the size of the runs recorded on the performance. Also, think about what implementing this as a Doubly Linked List would do and how the specified structuring would affect results.

1.2 Submission

Include answers to the theory questions in a comment at the top of your `StructSort.java` file, including the question you are answering. To submit, zip the source folder in your eclipse directory and call it `40254326.zip`.

1.3 Evaluation

In the zip file containing this assignment, you'll find additional files, notably `DebugRunner.class` and `config.xml`. The zip also includes several test inputs. Once you have a `StructSort.class` file, compiled from your written Java source, you can run `java DebugRunner` from the folder at the base of its package structure to receive feedback. This requires command-line operation. Please seek assistance from your tutors if you're unsure how to proceed. Most assignments will be accompanied by this setup. The config.xml file will grade your assignment partially so you can gauge your progress. However, remember that the final marking will be based on a more comprehensive `config.xml`. Hence, it's crucial to follow assignment instructions meticulously.

Structured Insertion Sort Example

You might type: *java comp352.assignment.two.StructSort inputs1.txt*

`inputs1.txt` might look like:

```
12 35 13 56 79 95 73 87 20 66 48 18 36 99 29
```

The output would look something like:

```
12 35 13 56 79 95 73 87 20 66 48 18 36 99 29
We sorted in DESC order
We counted 1 DESC run of length 2
We performed 5 reversals of runs in INCR order
We performed 14 compares during structuring
We performed 75 compares overall
We performed 55 swaps overall
99 95 87 79 73 66 56 48 36 35 29 20 18 13 12
```

Q2) Theory: Qualify This Structuring

How did the structuring pass you performed, specifically the reversals chosen, affect swaps and comparisons? Was anything else affected? Answer in less than 100 words.

Q3) Theory: Size of Runs

How do you feel the size of the specific runs you recorded (*DESCENDING* order of length 2) impacted performance? Answer in less than 100 words.

Q4) Theory: Doubly Linked Lists

What would implementing this as a Doubly Linked List do? How would the specified structuring affect results? Answer in less than 100 words.