Sort Comparison Documentation

# Description

The assignment was to analyze various sorting algorithms in Java and create an analysis report. The code for insertion sort, selection sort, quicksort, heap sort, radix sort, and merge sort were provided. Each contained a static method that performed an in-place sort on a static list of primitive ints or wrapped Integers given as a parameter. A table was also given containing blank entries for the experimental results of each sort tested against four different array types: sorted in order, sorted reverse order, almost sorted, and randomly sorted. Each sort’s comparisons, movements, and CPU time were expected to be recorded.

I modified these files to extend an abstract class Sorter that wraps the class and counts comparisons and movements. The Sorter class also contains an abstract method ‘sort’ that performs that class’ sort. The ComparisonRunner class is where all the calculation is done for each sort and contains the main method. When main is called, a ComparisonRunnerGUI object is instantiated, setting up the GUI. It collects input from the sliders and sends them to ComparisonRunner when the ‘Run test’ button is pressed. Random arrays are created by ArrayMaker according to user specifications and are used by ComparisonRunner to test each sort. Finally, the results are displayed on table in the GUI.

# Input

No external input files are used. All input is received from the user via the GUI. The list length, number of trials, and ratio of sorted to unsorted elements in the almost sorted list can be adjusted using sliders or entering a number. Sorts can be selected to be tested or untested using toggleable buttons.

# Output

No external output files are used. All output is given from the table in the GUI, including CPU time in nanoseconds, number of comparisons and movements, and the percent difference between the winner and all other sorts for each list type.

# Overview of Data Management and Flow

The ComparisonRunnerGUI constructor stores list length (int), number of trials (int), ratio of sorted elements (double), and selected sorts (static Boolean list) for use in ComparisonRunner.groupSortTester(). This method is called four times (for each list type) and returns four 2D arrays of longs containing the CPU time, comparisons, and movements of each sort. Static Double lists are created and filled with percent differences for each list type, as well as longs for each list type’s winner time. Finally, all the data is displayed on the table.

# Specific Method Explanations

## ComparisonRunnerGUI

ComparisonRunnerGUI

First, it instantiates frame, panels, labels (static display text), text fields (editable text fields), buttons, sliders, and a table. The GUI is organized so that the overall frame houses a main panel, which breaks into three different panels: grid, input, and button. The grid panel contains the table that displays output, the input panel contains all sliders and the ‘Run test’ button, and the button panel contains buttons that toggle which sorts to exclude from testing. The input panel was set as a GroupLayout (as opposed to a BoxLayout or GridLayout) which is more variable than other layouts, but requires more setup.

A horizontal group must be set to manage the bounds of the horizontal axis, while a vertical group is set to manage the bounds of the vertical axis. Additionally, both horizontal and vertical groups can be sequential or parallel. A sequential group along the horizontal axis organizes components in a simple horizontal line and requires a matching parallel group along the vertical axis to be correctly set. Inversely, a parallel group along a horizontal axis must also have a corresponding sequential group along the vertical axis.

Once the input panel setup is completed, the three panels are added to the main panel. Finally, the main panel is added to the frame and the frame is set to visible. However, one thing I skipped over were the buttons. Each button needs an action listener to be added to it, which is a method that can be declared in the arguments of a button object’s addActionListener method. The action listener of the ‘Run test’ button takes the current input values of the GUI and uses the commented out section in the main method to perform the sorts. The output lists are displayed on the table using some convoluted mess of spaghetti code that I currently can’t even decipher. All I know is that it accounts for the gaps in the table between each list type section, the reserved slots in the first row and column, and the boundary of the table itself.

## ArrayMaker

getAlmostOrderList

Almost-sorted lists are lists with scattered ‘fragments’ of sorted elements, so I predetermined which indices will be sorted and left the rest unsorted. The number of sorted elements is the list length multiplied by the ratio given as a parameter to the method. Randomly selected indices to be sorted are stored in a HashSet and while filling up the list, the current index is checked against this set. If the set contains the index, the value at this index will be set as the value itself. Otherwise, a random unused number not part of the set of selected indices is set as the value. This value is added to another set that keeps track of numbers randomly picked and already used to prevent duplicate random values.

## ComparisonRunner

individualSortTester

Parameter ‘type’ is a number from zero to five that indicates which sort to test. An abstract sorter class is instantiated with a specific sort object and calls ‘sort’ using polymorphism. A long stores the time taken by the system (in nanoseconds) to complete the sort call and is placed in an output array. The number of comparisons and movements calculated during the sort is retrieved from the sorter class and placed along with the CPU time in the output array, which is returned.

groupSortTester

This method synthesizes all the data from every individual sort returns a two-dimensional array with data type long. The 2D array is six rows long (one for each sort) and three columns wide (time, comparisons, movements). A loop is entered and goes on for however many trials are specified in the method parameters. At the end of the loop, the average over all trials is found by dividing each cell by the number of trials. During the loop, a new list of a list type (in-order, reverse-order, almost-order, random-order) given by method parameters is made before deciding whether to test each sort. A static Boolean list given by the method parameters indicates whether a sort should be tested and is checked before collecting data on the sort. An empty list is made for untested sorts. At the end of the loop, each sort’s data is added to the 2D array and iterates. At the end of the method, the 2D array is returned. Remember, the data returned is only for a specific list type, not all four list types combined.

getWinningSort

This method takes the data from groupSortTester and returns a static Double list that contains the percent difference between the wining sort and other sorts. The minimum time among the sorts is found and set as a reference for the calculation of percent difference between it and the others. Each percent difference is then rounded to a single decimal and added to an output array. A value is added to the end of the array that contains the index of the winner to save time searching for it in later methods. The output array is then returned.