Lab 7 – Sorting and Searching

due Oct 4th at 23:59

Objective

Today's lab will help you get familiar with

- The composite design pattern
- Sorting and stable sorting
- Binary search

The Composite Design Pattern:

The composite design pattern is a generalized pairing of two data, combined in such a way to allow the pair to be handled as a single unified object. In Java, this may be implemented by two fields in a class with appropriate accompanying methods. See Section 3.9 for more details.

Stable Sorting:

When a complex object like a pair is sorted based on only one of its fields, there are different possible outcomes of the sort. The order of objects which match in the sorted field is unspecified. It is often useful to specify this order further. One such specification is to require that in the case of such a tie, the original ordering of the objects prior to the sorting is preserved. A sorting algorithm is said to be stable if it satisfies this extra condition.

Binary Search:

In this lab you will need to implement a binary search method. This method provides an efficient way to search an array for a value given that the array is sorted. If you have not gone over binary search in class the following provides an explanation of how it works:

https://www.tutorialspoint.com/data_structures_algorithms/binary_search_algorithm.htm

---

Assignment:

1. Create a class Pair which stores two variables: key and value. Both of these fields should be generic implementing Comparable. Pair should have constructors, setters, getters, and a toString() method which relies on the toString() methods of the fields. (Every class in Java inherits a toString() method from the Object class, if this is not overridden elsewhere!)
2. Create a MyArrayList class extending ArrayList of Pair. MyArrayList should also have a toString() method, a quicksort() method implementing quicksort as in Figure 8.22, a stableQuicksort() method implementing a stable adaptation of quicksort (if , and a binarySearch() method implementing binary search for a given value, returning the first matching Pair or null if no matching pair is found. Use the following algorithm for stableQuicksort(): Create an auxilliary MyArrayList of Pairs; each Pair is to contain the initial position in the old MyArrayList as key and the old value as value. Then sort the array; if two Pairs have identical values, use the key to break the tie. After the new MyArrayList has been sorted, rearrange the original MyArrayList accordingly.
3. Your classes should be fully unit tested. In particular, you should have at least one test case demonstrating that stableQuicksort() provides a stable sort of some input for which quicksort() does not. Have these tests display pre and post sorting lists and include a screenshot of this in your lab notes.

Submission:

In addition to your code you must submit lab notes(in PDF format). Save your lab notes in the project folder before you compress it and upload it. Details about the lab notes can be found on the CS150 lab Moodle page.

Grading:

1. Pair implementation - 2
2. MyArrayList implementation - 4
3. Unit testing- 2
4. Style/commenting – 1
5. Lab notes - 1