Timing Sorts

Lab 3

Tuesday, February 7, 2012

CSC 062: Spring, 2012

In this lab, we'll be playing with some of the sorting algorithms we've discussed in class. In addition, you'll get some familiarity with the `merge` method of `MergeSort`, which you'll need for your next assignment when you implement an on-disk version of the algorithm.

You may again work in pairs on this lab, but choose a partner that you have not worked with before.

Getting started

Create a new project in `Eclipse` and then go to `Terminal` and copy over all the ".java" files from `/common/cs/cs062/labs/lab3`.

To tell the "cp" command that you only want ".java" files, append `/*.java` after the directory name. '*' is a wildcard that says match anything, so you will get all files in the above directory that end in ".java".

After you've copied the code, spend 5 minutes looking at the different classes. In particular,

- Look at the interface `Sorter`.
- Look at how the `Quicksort` and `MergeSort` classes implement the interface
- Look at how the `main` method of the `SortTimer` class is able to print out data for an arbitrary number of `Sorter` classes (this is the benefit of using an interface!).
- Notice that the `SortTimer` class does a check for correctness after sorting. If you make a mistake in implementing your `merge` method, you'll get an error here.

Finish `MergeSort`

I've given you all of the code for this lab except the `merge` method, which you should now implement. Give it a good effort, but if you get stuck, I've provided a solution below. It will benefit you to figure it out in the lab, though, while you have help from me (i.e. without looking) since you will be implementing something similar for your assignment.

Once this is done, you should be able to run the `SortTimer` class.

Play with the timing

Notice that we have called the printTimes method twice in the `main` method of the `SortTimer` object. Run the `SortTimer` class. What explains the very different answers obtained in the two runs for small values of size? Does the data obtained from the second run look like you'd expect? Which one is faster?

This should give you some confidence that `Quicksort` average case works as we expect. As an additional test, change the `printTimes` method to generate sorted data instead of random data. For example, have it fill the array with the numbers from 1 to size. How does this change your timing data? Is this what you expected?

Playing with the sorting algorithms

Open a terminal window and type the following command:

```
/common/cs/cs062/labs/lab3/bin/coinsorter
```

You will see a window similar to the one for the Silver Dollar Game, except that all the squares are filled, and the coins have different sizes. Use the keystrokes below to shuffle and sort the coins. Experiment with several of the sorting algorithms.

`c`: sort the coins using a randomly-selected algorithm

`i`: sort the coins using insertion sort

`q`: sort the coins using quicksort

`r`: rearrange the coins into a random order

`s`: sort the coins using selection sort

`x`: exit the program

The program you are using has a few additional features. Typing `f` (for "freeze") stops the sorting; typing `t` (for "thaw") resumes the sorting. Typing `f` when the sorting is frozen advances the algorithm by one step. You can continue to type `f` to proceed step-by-step, or `t` to resume normal execution.

Typing `c` selects one of the sorting algorithms at random and executes it. Practice with the `c` command to develop your skill in identifying the algorithm from the pattern of comparisons and swaps.

If you still have time...

Implement a new class for one of the $O(n^2)$ running time sorting methods that `extends` our `Sorter` interface. Add this new class into the `SortTimer` class and compare its runtime to the other sorting methods.

Submitting results

Write a brief summary of the work you did today, and the results that you received. For example, do you get different results from the same algorithm for small size lists? How about larger lists? Do some algorithms seem much faster than others in the average case? In worst case? Feel free to comment on anything that you learned, or anything that was interesting.

You will need to submit a text file with your summary to the dropbox folder. If you have chosen to work in pairs, then your file should be named something like "Lastname1-Lastname2-Lab2.txt" where your lastnames are used in the filename. Remember not to use spaces in the filename.

An implementation of `merge`.

Here is one implementation of the `merge` algorithm. It uses an extra ArrayList, and so mergesort does not sort "in place" as our other algorithms do.

```
/**
 * Merge data >= low and < high into sorted data.  Data >= low and < mid are in sorted order.
 * Data >= mid and < high are also in sorted order
 *
 * @param data the partially sorted data
 * @param low bottom index of the data to be merged
 * @param mid midpoint of the data to be merged
 * @param high end of the data to be merged (exclusive)
 */
public void merge(ArrayList<E> data, int low, int mid, int high){
        ArrayList<E> temp = new ArrayList<E>(high-low);

        int lowIndex = low;
        int midIndex = mid;

        while( lowIndex < mid &&
                   midIndex < high ){
                if( data.get(lowIndex).compareTo(data.get(midIndex)) < 1 ){
                        temp.add(data.get(lowIndex));
                        lowIndex++;
                }else{
                        temp.add(data.get(midIndex));
                        midIndex++;
                }
        }

        // copy over the remaining data on the low to mid side if there
        // is some remaining.
        while( lowIndex < mid ){
                temp.add(data.get(lowIndex));
                lowIndex++;
        }

        // copy over the remaining data on the mid to high side if there
        // is some remaining.  Only one of these two while loops should
        // actually execute
        while( midIndex < high ){
                temp.add(data.get(midIndex));
                midIndex++;
        }

        // copy the data back from temp to list
        for( int i = 0; i < temp.size(); i++ ){
                data.set(i+low, temp.get(i));
        }
}
```