CSCI-1302

Project 5

Software Design
Spring 2012 – University of Georgia

Sorting out Sorting!

Goals	1) Implement insertionsort, bubblesort, happyhoursort, and quicksort 2) Implement a project using recursion via QuickSort 3) Gain experience developing testing programs 4) Design, implement, and evaluate a study. 5) Use Javadoc
Points	This project is worth 75 points
Due Date	This project is assigned on April 5, 2012 and is due on April 19, 2012 at 11:00 PM
Late Penalty	You may submit this assignment for up to three days past the due date, where each 24-hour period counts for a 12% deduction off of the original value.
Collaboration Policy	For this assignment, you may not collaborate with any one other than the teaching assistants and instructors. You may post questions—not code—on Piazza.

Project Details

Each team will implement and analyze the average (amortized) performance of four sorting algorithms: **insertionsort, bubblesort, happyhoursort, and quicksort**. We've studied insertionsort and quicksort in class and the algorithms may be found in the Gray text book.

What is happyhoursort? You might've noticed during the Sorting Out Sorting video there was a sorting algorithm called **bubblesort** that had really poor performance. Bubblesort works by repeatedly going through a list and comparing adjacent items and swapping them if they are in the wrong position. You're done with the bubblesort if you pass through the list of elements and no swaps are needed. From Sorting Out Sorting, it was clear that smaller elements would "bubble" up the list, and larger elements would "bubble" down the list.

Happyhoursort, or the cocktailsort**,** is a variation of bubblesort. It differs by how the algorithm goes through the list. In bubblesort, you repeatedly scan in one direction, while in happyhoursort, you first scan left-to-right, then right-to-left.

To implement happyhoursort, you'll need to use three components: a while loop that contains two for-loops in series. The first for loop will scan left-to-right, while the second for loop will scan right-to-left. The while loop's conditional is on whether or not no-swaps have been made—remember, if you aren't swapping any more items, your list is sorted! But, think carefully if the while conditional is the only time you'd want to "exit" the sorting process.

Implementing Sorting.java

Your team, and by team I mean you, will implement a class called Sorting.java that contains the implementations of the three sort algorithms by creating four public static methods respectively named **insertionSort, bubbleSort, happyHourSort, and quickSort**. To simplify this project, we're going to constrain the algorithms to just sorting arrays of ints. Each of the methods of Sorting is required to take an array of integers (the primitive data type int) as a parameter and sort the array in ascending order. Each of these methods will then return a sorted array of the primitive data type int.

Implementing SortExperiment.java

You will also implement a driver class called `SortExperiment` that contains a main method. You'll use this class to conduct some experiments comparing the three sorting algorithms.

- `SortExperiment` should generate 30 different arrays of `ints`. These should contain a randomly-generated set of integers. The size of these arrays should also be determined experimentally and be bounded between 10 and Y elements. You will need to experimentally determine what a "good" value of Y is where your program 1) finishes execution in a reasonable manner (i.e. no more than 3-5 minutes), and 2) you can see some sort of performance distinction between the various sorting algorithms. Your justification for the Y you choose should be explicitly explained in your writeup.
- `SortExperiment` should also implement a method called `verifySort` that verifies whether an array is actually sorted. Your method should have a runtime of O(N) where N is the length of the length of the input array. You should provide some visible feedback that things are hunky dory.
- `SortExperiment` also must measure the running time of each sorting algorithm used on each sample. In order to measure the time for each algorithm, you can use the `System.currentTimeMillis`() method that returns the time in milliseconds. For example:

  ```
  long startTime = System.currentTimeMillis();
  somevalue = callAMethod(param);
  long runtime = System.currentTimeMillis() - startTime;
  ```
- Your program must use Swing to plot the data that you've collected. This is a critical feature of this project; you must think about how to adjust any scaling (as needed) on your output. You should also follow good graphing techniques, using labels, ticks, titles, etc., as appropriate.

Important Note: Nike is a multi-user AND multi-tasking system, and thus there are many factors that influence timings (such as number of users, tasks being run, etc). When you're on Nike (or any machine that allows more than one user to sign on, you can also type the "who" command to see if someone else is remotely logged onto that particular machine.

Your Goals

1) Your goal is to create some hypothesis about the performance of these various sorting algorithms for various Ns while on a standalone system (i.e. a Mac in the 307 cluster or your own personal computer) while also using a multi-user system such as Nike. Submit your hypotheses by **Thursday, April 12**, by 11PM as a PDF document on eLC. You'll use these hypotheses again in your formal writeup.
2) Implement the sorting algorithms as defined above.

3) Run a suite of test cases through the different algorithms to get timings to use in your plots
4) Write up a formal experiment document reporting your results.

Reporting the Results

You also need to turn in a formal writeup of this programming project, maximum of 5 pages single-spaced using an 11-point serif font. Your report will contain the following sections:

1) Title Page with your name and email address, and a title (that does not count towards your page limit!)
2) Experimental design description, detailing the high-level nature in how your experiment testing operates
3) Hypotheses: you should state that you believe the outcome of your experiments should be, based on the theoretical time complexity of the three sorting algorithms
4) Results: using a graph or graphs comparing the performance of the three algorithms on the random samples.
5) Discussion of the results indicating when you'd want to use each of the sorting algorithms, and comparing your results with your hypotheses and explain (or suggest reasons) for any differences. You should consider the cost versus benefits of various algorithms.

Point Distribution

Proper documentation 25 points

All methods, classes, constructors, and class variables must be commented in the Javadoc format. The makefile must create the Javadoc files in a folder called javadoc for all private, public, and protected methods, constructors, classes, and class variables. Parameters, return values, and exceptions must be included in the Javadoc for each method and constructor. All classes must include the name (first and last) of the author of the project

Test Cases 10 points

Passing test cases and properly handling "interesting" input, which includes (but is not limited to) files that do not exist, incorrect command line arguments.

Quality of Graph 15 points

Experimental Writeup 25 points

Total: 75 points

Delivery Instructions

Science!

1. Create a folder in an Nike account called **lastname1 _proj5** where lastname1 is your lastname.
2. Copy all **thoroughly commented** Java source files in the folder created in step 1.
3. Place a working makefile in the folder created in step 1 that has three directives:
   a. compile: compiles all of the source code and creates Javadoc
   b. run: runs an example of your program
   c. clean: removes all class files

4. Add a readme file to the folder created in step 1 which has your name and contact information as well as clear instructions on how to compile and run your team’s program.
5. Remove all class files before submitting.
6. Navigate to the parent directory of the folder created in step 1 on Nike, and issue the command below.

   ```
   submit  lastname1_lastname2_pro5  cs1302a
   ```

7. If the submission was successful, then a file that begins with rec will be created in the submitted folder.