UNIVERSITY OF VIRGINIA, DEPARTMENT OF COMPUTER SCIENCE

Lab 07 - Sorting

Dr. Mark R. Floryan

October 7, 2018

1 PRE-LAB

This week, you will be implementing four sorting algorithms. For the post-lab section, you will be analyzing the performance of these sorting methods.

1. Download the provided starter code.
2. Implement four sorting algorithms.
3. Use the provided tester to check if each methods are working correctly.
4. **FILES TO DOWNLOAD:** sorting.zip
5. **FILES TO SUBMIT:** lab07.zip

1.1 SORTINGALGORITHMS.JAVA

You will be implementing four sorting algorithms for this lab. They are listed below:

```
/**
 * Slow sort number 1. Bubble sort.
 */
```

```java
public static<T extends Comparable<T>> void bubbleSort(T[] list);

/**
  * Usually slow sort number 2. Insertion sort.
  */
public static<T extends Comparable<T>> void insertionSort(T[] list);

/**
  * Recursive merge sort and associated private helper method
  * the second method below provides the portion of the array
  * (i.e., index i to j inclusive) that we want to sort.
  */
public static<T extends Comparable<T>> void mergeSort(T[] list);
private static<T extends Comparable<T>> void
                                mergeSort(T[] list, int i, int j);

/**
  * Recursive quick sort and associated private helper method
  * the second method below provides the portion of the array
  * (i.e., index i to j inclusive) that we want to sort.
  */
public static<T extends Comparable<T>> void quickSort(T[] list);
private static<T extends Comparable<T>> void
                                quickSort(T[] list, int i, int j);
```

You may add additional helper methods that might be useful in implementing these sorting algorithms. Some methods that may be useful, but are optional, include swapping two elements given indices i and j, merging two sorted lists into a single sorted list, partitioning a list around a pivot, etc.

The provided tester (main method) will call each of your sorting methods one at a time. Above the main method is a variable you can change to increase or decrease the size of the lists to sort while testing. The tester will call each of your four sorting methods one at a time. For each, it will take the result and check if 1) the sizes of the unsorted and sorted lists are still the same, 2) the elements in the sorted list are indeed in sorted order, and 3) the sorted list contains the same elements in total that the original list does. If any of these checks fail, you should get a notification to that effect.

2 In-Lab

The goal of this in-lab is to continue practicing with sorting in a laid back environment. As usual, you will:

1. Get into small groups of two
2. The TAs will present you with some programming challenges regarding trees. These will not be graded, but you are required to attend and to participate.
3. You may think about the challenges before lab below if you'd like, but we highly recommend that you not solve them ahead of time.
4. The TAs will give you time to solve each problem and lead you in sharing solutions with one another.

2.1 Sorting

The TAs will lead you in going through the following challenges. It is ok if you do not get through each of these.

1. Re-write insertion sort, but have it sort a linked list (given as a parameter) instead of an array.
2. Update your quicksort implementation to use the median of medians algorithm found here. The median of medians is used to select which item in the list to use as your pivot. Then, call quicksort on a reverse sorted list. How much faster does the algorithm get?
3. Implement merge sort iteratively (i.e., no recursive calls allowed). You can still invoke your merge method within your iterative loops.
4. **Bonus Challenge**: Implement merge sort in-place. This means that the algorithm can only use $\Theta(1)$ extra space. We will not include the recursive space necessary to make the recursive calls. Thus, this challenge reduces to the following: Can you implement the merge subroutine without allocating the extra array?

If you have coded up all of these and they work, then show the TA and you may leave early.

3 Post-Lab

The goal of this post-lab is to produce a report analyzing the strengths and weaknesses of various sorting algorithms.

You will perform some experiment(s) by doing the following:

1. Experiment 1: Run an experiment in which you fill an array with random values between 1 and 1,000,000. Time each sorting algorithm. Make sure the list is large enough that comparing the runtimes of the algorithms shows a meaningful difference.

2. Experiment 2: Run each algorithm on a large list of elements that begins in reverse sorted order. How do these results compare with that of experiment 1?

3. Experiment 3: Run each algorithm on a list that is "almost" sorted. Here, by "almost sorted", we mean that each element in the unsorted list is about 5-10 positions away from its correct position, but almost all elements are not in their final position.

4. Experiment 4: Update your code to add a "hybrid sort" that switches algorithms mid-executions. Design this algorithm such that you take advantage of the results obtained in experiments 1-3. Then, go back and run experiments 1-3 again with your new algorithm. Can you get yoru algorithm to beat all of the others in every experiment?

5. **FILES TO DOWNLOAD:** None

6. **FILES TO SUBMIT:** PostLabSeven.pdf

For each experiment, make sure that you are sorting large enough lists such that differences in the algorithm speeds begins to show.

3.1 Report

Summarize your experiment and your findings in a report. Make sure to adhere to these general guidelines:

- Your submission MUST BE a pdf document. You will receive a zero if it is not.

- Your document MUST be presented as if submitted to a professional publication outlet. You can use the template posted in the course repository or follow Springer's guidelines for conference proceedings.

- You should write your report as if it is original novel research.
- The grammar / spelling / professionalism of this document should be sound.
- When possible, do not use the first person. Instead of "I ran the code 60 times", use "The code was executed 60 times...".

In addition to the general guidelines above, please follow the following rough outline for your paper:

- **Abstract**: Summarize the entire document in a single paragraph
- **Introduction**: Present the problem, and provide details regarding the algorithms you implemented (especially the "hybrid" algorithm).
- **Methods**: Describe your methodology for collecting data. How many method calls, how many executions, how you averaged things, how large were the lists, etc.
- **Results**: Describe your results from your execution runs.
- **Conclusion**: Interpret your results. Which algorithm was fastest in each situation? Did the fastest algorithm change? If so, why? Does the performance you see match the theoretical runtimes of the algorithms? Why or why not?

Lastly, your paper MUST contain the following things:

- A table (methods section) summarizing the experiments and how many execution runs were done in each group.
- At least one table (results section) summarizing the results of all of your experiments.
- Some kind of graph visualizing the results of the table from the previous bullet.