CS 312
Assignment #7
Sort Fight at Midnight

Due 11/16/18, in class

1. Goal
The goal of this assignment is to empirically study several sorting algorithms. And, of course, to help host this year’s annual Bikini Bottom sort fight. Here is who has signed up so far — Patrick with his bubble sort, Squidward with his selection sort, Mr. Krabs with his insertion sort, Sponge Bob with his quick sort, Sandy with her tree sort, Plankton with his heap sort, and Gary with his merge sort.

2. Problem Statement [client’s statement of their need]
The Bikini Bottom Civic Association (BBCA) needs a list generator and a sort program for each contestant registered in this year’s sort fight. The list generator will generate a list of random numbers for each sort to put in sorted order.

3. Requirements Analysis [What is the client’s problem?]
Q: What are the input, arguments, and output of the generator?
A: The generator reads no input, but does take two command-line arguments: a seed and a count. It prints (to standard out) count random integers (one per line) after seeding the random number generator with the seed.

Q: What does each contestant program need to do?
A: Each contestant’s program will read in a list of numbers from standard in, sort them, and then output the sorted list (one per line) to standard out.

Q: What are the requirements for the competition?
A: The competition has two requirements: correctness and timing. For correctness you must verify that each contestant’s output is sorted using the Linux command sort. For the timing use the program /usr/bin/time to output the time taken by each sort.

4. Design [How]
[ This I’ll leave up to you. I’m happy to look at your design and corresponding build plan if you like. FYI, one design decision I made was to have generator first output the number of ints. Another was to have an abstract class ASort that all the sorts extend. You are welcome to do the same, but not required to. ]

What to hand in
[ Optional: By 6am Wednesday push a 1-2 paragraph summary of the requirements and any questions to README.md. I will comment on however much you have pushed. ]

(1) A well-formatted printout of empirical-study.pdf, your empirical study of the timing data.

(2) A GitHub repo that includes (you must use these names as the grading script assumes them!)
  • README.md with the sections Problem Description and Cheater Check (see below)
  • empirical-study.pdf
  • your Java source code
  • a spreadsheet with the two tables (an example table is given below).
    Consider adding a chart or two.
    Consider using the Linux spreadsheet program soffice.
Assignment Requirements  [ part of being a course rather than software development ]

- All sorts must be verified correct. Correctness is checked using the Linux `sort` command as follows
  ```
  java generator 42 100 | tail -n +2 | sort > oracle.out
  java generator 42 100 | java PatricksBubbleSort > patrick.out
  diff -s oracle.out patrick.out
  ```
  Note that I used “| tail -n +2” to remove the first line of Generator’s output, which is the number of ints.
- To collect the timing data replace `java PatricksBubbleSort` with `/usr/bin/time java PatricksBubbleSort`.
- The competition! First, find a count that causes Patrick’s sort to take approximately ten seconds. Call this count \( C \). Then for the counts \( C, 2C, 4C, 8C, \) and \( 16C \) run each sort and record, in a separate table, the user and elapsed time for each run as output by `/usr/bin/time`. In addition, record that times obtained using the Linux `sort` command.

<table>
<thead>
<tr>
<th>User Time</th>
<th>Patrick</th>
<th>Squidward</th>
<th>Mr. Krabs</th>
<th>Sponge Bob</th>
<th>Gary</th>
<th>Sandy</th>
<th>Plankton</th>
<th>sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 2C )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 4C )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 8C )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 16C )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Sponge Bob says that “9”’s make Patrick’s sort faster and that’s cheating! He wants to prove it. Using `grep` (rather than editing your Java code) provide the command line that will run one of the sorts with only numbers that include at least one “9” digit. Then provide the opposite where you filter out numbers that include a “9” digit. Include these two command lines in your README.md under the heading ## Cheater Check. Was the Sponge right?
- Each source code file must start with
  ```
  /**
   * This is my code! It’s goal is to ....
   * @author Your Name
   * @version 1.0 9/18/2018 (a version number followed by the date)
  */
  ```

Notes
- While you will learn more if you code the sorts from scratch, check out http://www.java2novice.com/java-sorting-algorithms
- Also check out www.amazon.co.uk/Sinclair-2X-Spectrum-Vega-Console/dp/B013BC71R6
- Reflect on the relationship between the run times and the relative challenge (e.g., the number of bugs, time taken to code, code length, etc.) of coding the different sorts.