Computer Science 2

Spring 2018 – Dr. Gurka Project #8: Sorting

March 2 Specifications

# Description

Complete a program that compares basic sorting algorithms and analyze the results.

# Details

1. First, read and understand the specifications. Then create a project schedule for yourself (template posted on Moodle).
2. Read and understand the code. This will provide a lot of detail on how to do various parts of the project.
3. You may borrow code from the text for the sorts themselves; include the reference in your documentation. Make any changes needed to fit with the basic program handed out. Note that you are required to understand the sorts completely, and, once the project is complete, you must be able to code them yourself.
4. Sorts to use (5)
   1. N2 sorts: bubble sort (already done), insertion sort, selection sort
   2. N log N sorts: merge sort, quicksort
5. Use N = 10,000, N = 50,000, and N = 100,000 elements in the array to be sorted. If it so happens that your system is especially powerful, you may need to go to N = 1,000,000 to get useful results (where useful means pointing out efficiency differences); in that case, don’t output N = 50,000 results.
6. Count compares, swaps or shifts (insertion sort), and passes (N2) or recursive calls (N log N sorts); output to a file a well-organized report with data from all the sorts, at the three values of n ([10,000; 50,000; 100,000] or [10,000; 100,000; 1,000,000). Also output elapsed system time for each sort, at each N. Include a heading with your name, the course name and section, and the date. Be sure the timing is just the sort itself (not initialization, not copying the array, not output).
7. Don’t print out the large unsorted and sorted lists, but do be sure sorting worked right by calling the *isSorted* method after each sorting algorithm finishes. If it finds unsorted results, then your algorithm is flawed and needs to be corrected. For debugging use very small lists and examine (print) the entire list to see the problem.
8. Write the “best” version of each sort (for example, bubble sort stops early when the array becomes sorted and the passes do not extend into the sorted subarray).
9. Get one sort working correctly before starting the next. Get all the requirements implemented, tested, and double-checked before starting any extra credit.
10. Analysis discussion / report (a significant part of this project). These are your explanations, not material you found and copied.
    1. Comment on how the data shows that N log N sorts, as a group, are significantly different from N2 sorts; and how sorts within one big-O group compare to each other.
    2. Discuss any surprising or interesting results.
    3. Give best case, worst case, and average case for these algorithms and explain why the different data orderings make a difference (data-dependent) or why they don’t (data-independent).
    4. Discuss any other facts you discovered about sorting algorithms. Try already sorted data, data with lots of duplicate values, or some other variations of data). Try different versions of the algorithms, if you wish.
11. Scheduling discussion. Explain how closely you stuck to your original schedule and what you would do differently next time you plan a project.
12. Extra: Implement shaker sort as discussed in class; include results in your output, and discuss it in your report, including a specific comparison to bubble sort.
13. Extra: Write a discussion about using linked lists for one of the sorting algorithms. Would it be easier or harder or impossible to code? Would it be more or less costly in space and time? What changes would you need in the algorithm(s) (if any) to use linked lists? Would a doubly-linked or circularly-linked list be easier or better? Support your answers.
14. Extra: A bug report may be included on this project. Check the specifications on Moodle, and remember that you should begin documenting the bug when it appears, rather than trying to recreate what happened when you write the report.
15. No extra credit on projects with incomplete required components; no extra credit on projects with errors or with significant style or documentation problems (see the course standards); no extra credit on GOOJF projects. Extra credit must be done strictly on your own. Extra credit must be excellent to receive points.
16. Submission. Moodle: source code, output (all results, no full data lists), report as detailed above, any extra credit, cover letter with discussions. Drafts will be considered late submissions.

# Due Dates

* project schedule: Monday, March 5 / Tuesday, March 6, 2:00
* final project: Wednesday, March 14 / Thursday, March 15, 2:00 (last class day before break)