*CSC 505, Spring 2023 First Programming Project: Sorting*

**Due Sunday, February 19 at 9:00 PM**

### *Learning outcomes. Gain deeper understanding of (sorting) algorithms by implementing them in a non-traditional setting; explore the benefits and limitations of experimental analysis of algorithm behavior; demonstrate ability to manipulate linked lists; design experiments to test hypotheses about algorithm behavior; write a compelling comparative analysis based on experimental results.*

### Overview

You will implement and compare the performance of three sorting algorithms that sort *lists* (rather than arrays). These would be implemented as ***singly linked lists*** in Java (not the ArrayList class), C, or C++. A list can have a header record with pointers to the first and last cells. This assignment cannot be done in python for two reasons: (i) lists may be so large that runtimes for insertion sort are prohibitive; and (ii) python lists are manipulated like arrays.

The algorithms are insertion sort, merge sort, and Quicksort. You will compare the performance of these algorithms on random sequences of distinct integers, generated by [random\_permutation.py](https://drive.google.com/file/d/1XBon0wU_5ptSouCLstw_L7mGLj6nW250/view?usp=share_link), a program we provide. A key feature of this generator is that it allows you to generate sequences in blocks of equal size. So if the sequence is composed of blocks then, for each , all integers in are greater than those in .

The reason this feature is important is because insertion sort, with blocks of size , has worst case runtime rather than .

### Functional Requirements

Your programs must run on the command line and read a list of integers, one per line, from standard input. Output should be a sorted (in ascending order) list of those same integers, also one per line. Additional output giving statistics about the run should be sent to standard error. Format of the additional output is …

runtime SECONDS  
comparisons NUMBER\_OF\_COMPARISONS

The runtime should *not* include the time it takes to read the input or produce the output. All three programming languages have functions/methods that facilitate computation of runtime during execution. I have included classes/modules that handle this functionality in C, C++, and Java.

In addition to your three programs (and supporting modules) you should provide a simple shell script called run\_sort.sh. We will execute

| run\_sort.sh insertion < *file* | for insertion sort |
| --- | --- |
| run\_sort.sh merge < *file* | for merge sort |
| run\_sort.sh quick < *file* | for Quicksort |

Even though the generator produces instances with integers , your programs should work for any sequence of integers (all equal, just 0’s and 1’s, etc.).

### Constraints

*Your programs should not use any additional memory* except for a constant number of control variables and any (implicit) stack that handles recursion. This means that the lists must be ***relinked*** rather than copied. For example, where a recursive description in my notation says *x* + *L*, i.e., concatenating element *x* to the front of list *L*, the link associated with *x* must be redirected to the head of *L*. ***Do not create a new cell for x.*** The only part of your program that does memory allocation via **new** (C++ or Java) or **malloc** (C) should be where you read the input.

An interesting challenge that arises with linked-list implementations of merge sort and Quicksort is locating the *i*-th element of a list for a specific value of *i.* In the case of merge sort, you will need to locate the middle element: one clever way to do that is to use two pointers, one moving twice as fast as the other. For Quicksort you are required to either choose the pivot randomly or to use the median of first, middle, and last elements. On a side note, the list approach to Quicksort makes it easy to avoid including elements equal to the pivot in the recursive calls.

*Please make sure that the lists constructed from the input are in the same order as the input. This means that* ***new elements need to be appended to the rear of the list****.*Otherwise insertion sort will behave badly on nearly sorted inputs, the opposite of what is supposed to occur. And use self-documenting names for variables and functions/methods.

*Your code should be well documented with*

* comments that clearly indicate the relationship between each part of the algorithm and the segment of code that implements it
* compact methods/functions with names that clearly identify what they do
* descriptions of any parameters and/or return values
* descriptive variable names

Avoid detailed blow-by-blow comments on each line of code.

*It is acceptable to download or reuse (from other courses) code for basic linked list operations such as adding a new element to the rear, linking one to the front, and inserting an element into a sorted list. Code for the algorithms should be your own work.*

### Project report

A critical part of this assignment is a written report. The report should include the following elements.

**Experimental design.** Explain what hypotheses you tested or what research questions you explored. Describe the inputs you used (and explain why you used those inputs). Be sure that the set of inputs includes a variety of sizes and categories and some large enough to have runtimes of more than a minute for the slowest algorithm. Describe how many trials you performed for each size/type of input. Ignore trials with runtimes less than 0.05 seconds and round runtimes to the nearest 0.1 seconds.

**Experimental results.** Use charts and tables to effectively illustrate your experimental results. Provide an analysis of your results. What did they tell you about average case behavior of the algorithms? Did you observe anything unexpected?

Please see the [presentation about tables and charts](https://docs.google.com/presentation/d/1IMsgLX5ukem0Le4VRjZx9CVh1zXGuNXGzG3CrzDhenc/edit?usp=share_link) for guidance. The most common problems are (a) numbers that don’t line up (not right justified), (b) numbers that have too many decimal places (one or two should be enough); (c) large numbers without commas; (d) charts that have lots of curves on top of each other; (e) charts with only one curve; (f) labels on axes that are nonexistent or hard to read. Other issues and examples of good tables and charts are in the presentation (the bad charts are noted as such with explanations). You should spend some time cleaning up tables and charts. Don’t simply use unfiltered output from a program or script to create tables; if you use software that creates charts, play around with the options until the charts look good.

**Conclusions and future work.** Summarize your results and make some suggestions about future experiments that you believe are worth pursuing and why.

**Recommended Reading (also included in this folder).**

* [*How to Present a Paper on Experimental Work with Algorithms*](https://drive.google.com/file/d/1nXZhrrNKgsIwmplLMK6h4UqqNwbOhk0I/view?usp=sharing)*,* Catherine McGeogh and Berhard Moret, *SIGACT News,* November 1999.
* [*A Theoretician's Guide to the Experimental Analysis of Algorithms*,](https://drive.google.com/file/d/1mP7bXvn8Dh07bSo1Zeng1j6VCQZGSaEV/view?usp=sharing) David S. Johnson, in *Fifth and Sixth DIMACS Implementation Challenges,* American Mathematical Society.

### What to submit

*Moodle is set up for group submissions now. That means (I think) any member of your group can submit the required zip file. I’m not sure what that looks like on your end so you may want to do an early trial submission to check.*

Please submit a zip file that opens up a directory with a *distinctive name[[1]](#footnote-0)* containing

* all of your source code
* a README.txt file with simple instructions for compiling and running your programs
* the run\_sort.sh script specified above
* your project report as a pdf file
* a list of names and email addresses of all your team members including yourself in a file called collaborators.csv. Format is one line per team member, each line has the form name\_of\_person,email\_of\_person

### Division of Labor

I highly recommend that you ***not*** take the approach of assigning each person a set of tasks and then combining your efforts at the end. The most successful teamwork results if you cooperate closely on all tasks. Use pair programming during the implementation phase. Programming can easily be split into low-level functionality (methods that create lists from input, traverse lists, relink cells, etc.) and algorithm architecture (recursive calls, insertion, merging). Make heavy use of all communication platforms you feel comfortable with.

Discuss the experimental design and the main points of the report together. Certainly it makes sense to delegate a discrete task such as creating a single chart or table to one individual, but allow each person to do at least one pass through the text.

Above all, use the project as an opportunity to get to know each other and (I hope) enjoy working together.

### Expectations for groups of three

If you are in a group of three, I expect you to implement an algorithm that does merge sort or Quicksort and does insertion sort when the list size is ≤ *k* for some *k.* You can then do experiments with different choices of *k*. If you have other interesting ideas for additional experiments, let me know.

### Rubric

| ***Rubric item*** | ***points*** | ***comments*** |
| --- | --- | --- |
| **completeness** | 5 | are all the required elements present in the archive |
| **compilation and execution[[2]](#footnote-1)** | 10/alg = 30 | do the programs compile without error on a university vcl server running Linux and execute successfully on a variety of input instances |
| **code and comments** | 10 | is the code easy to read and easy to connect with the algorithm it implements |
| **experimental design** | 20 | are there coherent and well-considered hypotheses or research questions; is the set of instances chosen thoughtfully |
| **charts and tables** | 20 | are charts and tables compelling and easy to interpret |
| **analysis** | 10 | are the conclusions interesting, i.e., do they go beyond merely reporting what is easily seen in the charts and tables |
| **conclusions** | 5 | are there interesting suggestions for additional experiments |

1. For the directory name and all file names, use alphanumeric characters, underscores, dashes, and +’s only; no spaces please. In other words, make the names Linux-friendly. [↑](#footnote-ref-0)
2. If you are in a group of three, the three original algorithms are 8 points each and the algorithm that incorporates insertion sort into Quicksort or merge sort is worth 6 points. [↑](#footnote-ref-1)